NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIFORNIA

THESIS

DESIGN OF A HOVER MODE AUTOPILOT FOR THE PHOENIX AUTONOMOUS UNDERWATER VEHICLE

Juan Cesar Gonzalez, Jr.

June, 1995

Thesis Co-Advisors: Robert B. McGhee
Anthony J. Healey

REPORT DOCUMENTATION PAGE		Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE 1995 June	3. REPORT TYPE AND DATES COVERED Master's Thesis
4. TITLE AND SUBTITLE DESIGN OF A HOVER MODE AUTOPILOT FOR THE PHOENIX AUTONOMOUS UNDERWATER VEHICLE, UNCLASSIFIED		5. FUNDING NUMBERS
6. AUTHOR(S) Juan Cesar Gonzalez, Jr.		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey CA 93943-5000		8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.	12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)*

The potential uses for autonomous underwater vehicles (AUV's) is gaining recognition by organizations world wide. As such, continuous research toward improving existing vehicles by seeking new designs and increasing efficiency is being funded by organizations that serve to benefit from the new technology. Some examples for which AUV's may be used include mine countermeasures, submerged structural repair or destruction, search and rescue, biological study, ocean floor and coastal survey.

The goal of this thesis is to design an autopilot that will use a combination of both vertical thrusters, both horizontal thrusters, and the main screws simultaneously to control the NPS Phoenix AUV's posture during hovering conditions and short transits. The control design is implemented and simulated using Common LISP object oriented programming language. The results of this thesis are favorable. Since this thesis presents only a basic approach to an autopilot design, it is believed by the author that with further improvements to the design presented, the existing hover mode autopilot can eventually be upgraded with the resulting autopilot design. This upgrade would greatly increase the autopilots efficiency.

14. SUBJECT TERMS AUV Autopilot Design, AUV Simulation, NPS Phoenix AUV			15. NUMBER OF PAGES 121
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18 298-102

DESIGN OF A HOVER MODE AUTOPILOT FOR THE PHOENIX AUTONOMOUS UNDERWATER VEHICLE

Juan C. Gonzalez, Jr.
Lieutenant, United States Navy
B.S., Prairie View A & M University, 1989

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
June 1995

Accesion For	
NTIS CRA&I	☒
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution /	
Availability Codes	
Dist	Avail and / or Special
A-1	

Author: [illegible]
Juan C. Gonzalez, Jr.

Approved by: [illegible]
Robert B. McGhee, Thesis Co-Advisor

[illegible]
Anthony J. Healey, Thesis Co-Advisor

[illegible]
Michael B. Matthews, Second Reader

[illegible]
for Michael A. Morgan, Chairman
Department of Electrical and Computer Engineering

ABSTRACT

The potential uses for autonomous underwater vehicles (AUV's) is gaining recognition by organizations world wide. As such, continuous research toward improving existing vehicles by seeking new designs and increasing efficiency is being funded by organizations that serve to benefit from the new technology. Some examples for which AUV's may be used include mine countermeasures, submerged structural repair or destruction, search and rescue, biological study, ocean floor and coastal survey.

The goal of this thesis is to design an autopilot that will use a combination of both vertical thrusters, both horizontal thrusters, and the main screws simultaneously to control the NPS Phoenix AUV's posture during hovering conditions and short transits. The control design is implemented and simulated using Common LISP object oriented programming language. The results of this thesis are favorable. Since this thesis presents only a basic approach to an autopilot design, it is believed by the author that with further improvements to the design presented, the existing hover mode autopilot can eventually be upgraded with the resulting autopilot design. This upgrade would greatly increase the autopilots efficiency.

TABLE OF CONTENTS

ACKNOWLEDGMENTS

The research in this thesis was supported in part by the National Science Foundation under grant BCS-9306252.

I would like to express my most sincere appreciation to my thesis advisors, Dr. Robert B. McGhee and Dr. Anthony J. Healey, for their patience, friendship, and instructional guidance in assisting me to complete this thesis. I would like to thank the members of the NPS AUV group for making this project not only educational but enjoyable as well. Also, Robert H. Ellis, CDR, USN, for his time and professional assistance, and to the United States Navy, for providing me with this opportunity to improve myself so that I may better serve my country. Most of all, I would like to thank my lovely wife for her patience and understanding while I spent countless hours studying, her endless support and encouragement in my Naval career, and especially for her love and friendship.

I. INTRODUCTION

A. NPS PHOENIX AUV

The Naval Postgraduate School Phoenix Autonomous Underwater Vehicle (AUV) is a second generation autonomous underwater vehicle developed at the Naval Postgraduate School. Research in autonomous underwater vehicles at the Naval Postgraduate School began in 1987. The objective of this thesis is to design and simulate a hover mode autopilot that will use a combination of the two vertical thrusters, the two lateral thrusters and the main screws to control the Phoenix's posture during hovering and short transits when the low speed of the vehicle does not allow efficient use of the control surfaces. Linear control theory is used to design the autopilot based upon a derived mathematical model that represents the equations of motion. The design is implemented and simulated using LISP object oriented programming. The vehicle dynamics are modeled using the Newton-Euler three-dimensional dynamic equations. It is these equations that the autopilot responds to in order to determine the required thruster forces.

B. ORGANIZATION OF THESIS

This section presents the organization of this thesis. Brief descriptions of the chapter contents are discussed.

Chapter II introduces several different concepts being investigated by various research organizations. It also presents the current autopilot techniques implemented in the NPS Phoenix AUV.

Chapter III defines the specific problem addressed in the design of the autopilot. Boundaries surrounding the design and simulation of the hover mode autopilot are established and listed as initial assumptions and limitations.

Chapter IV documents the procedure followed in the design of the autopilot. The equations of motion are derived and provide the mathematical model for which the control equations are developed.

Chapter V discusses the underlying concepts in the simulation of the vehicle dynamics as well as the techniques for a pictorial representation of the vehicle's response the autopilot design.

Chapter VI presents the simulation results. Several observations are discussed. Chapter VII summarizes the thesis with recommendations for further study.

II. SURVEY OF PREVIOUS AND CONCURRENT WORK

A. INTRODUCTION

Autonomous underwater vehicles (AUV's) are gaining increased attention by research and development institutions world wide. The goal of this chapter is to inform the reader of ongoing AUV autopilot research projects by several different organizations. It will by no means be all inclusive. This chapter will begin with brief descriptions of different types of flight control autopilots; however, this thesis is concerned with the design of an autopilot for hovering, meaning control by use of thrusters and propellers only. The main difference between a flight control autopilot and a hovering autopilot is that a flight control autopilot can utilize control surfaces (fins and rudders) in conjunction with its forward motion propulsion system to control its attitude, whereas a hover mode autopilot cannot due to the lack of speed generated when the vehicle is trying to maintain a specific posture or short distance transits. A synopsis of the current autopilot implementation for the NPS Phoenix AUV will follow. Finally, discussion of previously developed software used for the simulation of the hover mode autopilot design is included.

B. BRIEF DESCRIPTION OF VARIOUS AUV MODELS

With the increased interest in AUV's, there have been many organizations that have designed and built AUV's for various missions (Yuh, 1994). The criteria necessary for a vehicle to be considered an AUV are that the vehicle be unmanned, untethered, and submersible. Hence the physical design has taken on various shapes, and since mission is not part of the criteria, most of the AUV's designed have been for a specific mission. In this section a brief description of some of the current AUV research will be presented.

1. NDRE-AUV

The Norwegian Defense Research Establishment (NDRE) designed, built and tested a flight control system for their autonomous underwater vehicle (NDRE-AUV) which was built primarily as a test platform for propulsion systems using seawater batteries (Jalving, 1994). The autopilot for the motion control was divided into three separate controllers, one each for speed, steering, and diving control. NDRE made their controller robust by applying redundant sensors and by increasing the margins of stability through design. In this vehicle, roll dynamics are neglected, and thus only five of the six degrees of freedom are controlled. It is discussed later how the NPS AUV also has separate controllers for flight control; however, Multivariable Sliding Mode Control concepts were used to incorporate robustness into the controller (Healey, 1993).

2. AQUA EXPLORER AE1000

The AQUA EXPLORER AE1000, a cable tracking AUV built in Japan by KDD R&D Laboratories, uses a fuzzy controller in their implementation of the autopilot (Kato, 1994). The fuzzy control concept involves the representation of control rules in a linguistic, qualitative manner. In the case of the AQUA EXPLORER, it required transforming each sensor input into five fuzzy variables negative big (NB), negative small (NS), zero (Z), positive small (PS) and positive big (PB). These five variables are then associated to each other by a set of fuzzy rules defined in (Kato, 1994). The control outputs are then obtained and applied to the thrusters and fins. All six degrees of freedom are controlled. The applicability of fuzzy control for a cable tracking AUV is quite feasible, since for its specific mission, boundaries for the AUV to operate within are required. ARPA also has developed and simulated fuzzy control for depth and pitch control (DeBitetto, 1994).

3. PTEROA

A Self-Organizing Neural-Net-Controller System (SONCS) is an adaptive motion control system that the Institute of Industrial Science, University of Tokyo has developed for use in their AUV called PTEROA (Fujii, 1990). The primary mission of the PTEROA is to take pictures of the sea bed. Although a Fuzzy Controller, as in the AQUA EXPLORER AE1000, is used to initialize the process of the controller, the algorithm used to implement the fuzzy logic is quite different. The neural network concept is used for its advantages of parallel processing and adaptability (or ability to learn). Neural networks also have been proven to provide fast, accurate solutions to nonlinear problems as is often found in control designs. However, neural networks are not always fast in convergence, and a PID controller can be designed to overcome this problem. From the results of testing, it is believed by the authors that highly reliable and robust control systems for AUV's are possible through use of the SONCS.

4. MUST

Martin Marietta Aero and Naval Systems have designed a flight control system based on Sliding Mode Control concepts and implemented it on their Mobile Undersea Systems Test (MUST) unmanned underwater vehicle (UUV) (Dougherty, 1990). The Sliding Mode Control concept was used in the design of both the transit flight control system and the hover flight control system. As in the NPS Phoenix AUV, hover is defined as the operating region where the thrusters and main screws are used for dynamic control. As part of the test for robustness of the hover flight control system, a vertical thruster failure was induced during a maneuver. The system response was favorable in that after an initial transient, the MUST UUV became stable with a constant error in pitch and depth, thus validating that Sliding Mode Control is robust in the face of control system uncertainties.

5. JASON ROV

The JASON ROV was developed by the DEEP SUBMERGENCE Laboratory of the Woods Hole Oceanographic Institute (Yoerger, 1986). Since this vehicle is an ROV (remotely operated vehicle) it requires an operator to maneuver it by controlling the vehicle with a joystick and monitoring the movement through a video camera mounted on the vehicle. A supervisory control system was designed for implementation using closed loop control based on Sliding Mode Control. This type of control simplifies the operators job in maneuvering the vehicle in that the operator only has to provide continuous commands for the desired trajectory while the system provides automatic control of vehicle position. While this is not exactly an autopilot since there is direct operator interaction, the control system presented can be utilized in designing an autopilot. Among the modes simulated for this vehicle were full trajectory control in body coordinates and earth coordinates. The full trajectory control in body coordinates is implemented using small angle displacements in which the coordinate system is constantly changing, that is, the "commands are defined in a coordinate frame that moves with the vehicle" (Yoerger, 1986). However, full trajectory control in earth coordinates provides a coordinate frame that is stationary allowing the operator to maneuver the vehicle more smoothly by viewing a map display vice the vehicle video. It is this latter type of control on which this thesis is based upon; the control in earth coordinates allows for large angle displacements about the fixed earth coordinate system as compared to the conventional small angle displacements associated with the body coordinate system.

The preceding discussion was presented to introduce the reader to several different control methods that are being studied and used by various organizations. Robust control is an important issue and required to satisfy the uncertainty in the vehicle dynamics. Although only a few examples were given, it is sufficient to say that the types of control methods introduced, Sliding Mode Control, Fuzzy Control and Neural Network Control, are all techniques that are being studied for the design of autopilots. Although it appears that no single method is better suited, the classical PID controller can perform well in both a flight control autopilot as well as a hovering autopilot and is used in the hover mode autopilot designed in this thesis.

C. AUV AUTOPILOTS AT NPS

The current autopilot design for the NPS Phoenix AUV is based on Multivariable Sliding Mode Control concepts (Healey, 1993). The Sliding Mode Control concept requires that sliding surfaces in the state error space be defined such that a sufficient relationship for the control element forces is obtained. This assures stability of the state variable errors and provides acceptable performance with closed loop conditions. The autopilot is divided into three separate subsystem autopilots which are: the Speed Control Autopilot, the Steering Autopilot, and the Diving Autopilot. The autopilot described in (Healey, 1993) is designed for transit mode, thus constant forward motion is required in order to maintain control of the AUV's position. More recently, hover mode control in a test tank environment using high frequency onboard sonar returns without the use of a transponder net to maintain both lateral and longitudinal position has been demonstrated (Torsiello, 1994). Details of modelling, sensor calibration, and control law design appeared in (Torsiello, 1994). The position control demonstrated longitudinal, lateral, and rotational control as separate, non-interacting control modes; however, the combined action of longitudinal, lateral, and rotational control is what needs to be explored.

D. PREVIOUSLY DEVELOPED SOFTWARE

All of the code used for the implementation and simulation of the hover mode autopilot design for this thesis is written in LISP. Some MATLAB code is used for ease of plotting the results of the simulation. A file named "robot-kinematics.cl" is used in its entirety without any modifications. This file contains functions that were developed by Professor McGhee (McGhee, 1994) to allow linear algebra manipulations in support of kinematic modeling. It also incorporates functions that perform homogenous transformations which are required for three-dimensional rotation and translation of objects. Another previously developed file (McGhee, 1994) used in its original form is "strobe-camera.cl". "Strobe-camera.cl" has functions that create a window for the object to be displayed. It uses the homogenous transformation matrix of the object to move the object within the window, hence creating the illusion of three-dimensional movement. Finally, some functions developed in the file "rigid-body.cl" are reused in a file named "auv-rigid-body.cl". These include a function to create a rigid-body class, implement the Newton-Euler equations, and integrate the Newton-Euler equations. The rest of the LISP code is original.

E. SUMMARY

As can be ascertained from the discussion in this chapter, research and development of AUV's is a relatively new field of interest that is being studied by organizations throughout the world. Recognition of its potential uses such as mine countermeasures, hydrographic and ocean floor mapping, submerged structure inspection, repair or destruction, marine life studies, hazardous material spill assessments, etc., will ensure continued and improved research (Yuh, 1995). NPS has taken an active role in this research arena since 1987 with emphasis on development of control technology for AUV's (Healey, 1993). This thesis provides the basis for further steps in this direction by designing and simulating a hover mode autopilot and investigating the use of LISP as a simulation language for numerical simulations of autopilots. The following chapters will discuss in detail the problem addressed, the design procedure, and evaluation of the results obtained.

III. DETAILED PROBLEM DESCRIPTION

A. INTRODUCTION

The objective of this thesis is to design a hover mode autopilot for the NPS Phoenix AUV and simulate vehicle response using Common LISP object oriented programming. The purpose of this chapter is to provide a description of the specific problem addressed in the design of the hover mode autopilot. This chapter begins with a detailed problem statement followed by a discussion of the assumptions and limitations for which the hover mode autopilot controller is designed.

B. DETAILED PROBLEM STATEMENT

The goal of this thesis is to design an autopilot for controlling azimuth, elevation, forward displacement, lateral displacement, and depth of the AUV using only the two vertical thrusters, the two lateral thrusters, and the main screws. Note that there is no attempt to actively control roll. This is allowed since the AUV was designed with a high degree of roll stability when submerged by using the natural roll righting moment as the correcting force and the hydrodynamic effects of the medium as the damping factor. Thus, there are six degrees of freedom, but only five are actively controlled. The actual posture of the AUV is to be determined by the navigation system where it is assumed that the position x, y, z, and the attitude as measured by the global Euler angles, with the exception of roll, are available as sensed data. For the purpose of the design and simulation, the Newton-Euler dynamic equations are used to determine the translational and rotational velocities and positions (posture). The autopilot acts to compare the actual posture with the desired posture. Any differences between the actual posture and desired posture are used to determine the thruster forces determined by Global Frame sensed data applied in the local body fixed frame to position the AUV until the actual posture and desired posture are equal. The following section describes the limitations of the design which are essential to setting bounds for this thesis. This work is contrasted to other work in multidimensional positioning (e.g. Offshore drilling vessels) where the departure of angular positions from a nominal position is small.

C. DESIGN PARAMETERS

This section discusses the design environment for which the hover mode autopilot controller is designed. It emphasizes the definition of hover mode and identifies the assumptions and restrictions utilized for the initial autopilot design.

As previously stated, hover mode control is control by use of thrusters and main screws only. To elaborate, hover mode includes those periods of dynamic control when the AUV is trying to maintain a specific posture relative to a fixed earth coordinate system. It also includes short transits where the AUV's speed will not be sufficient for the fins to be used as control surfaces such as in transit mode.

For the initial design of the hover mode autopilot controller, several assumptions and restrictions are imposed. This is done to set the basic foundation for the current design and so that future work will have a foundation to build upon. The assumptions used are as follows:

- Angular rate sensors are perfect.
- Accelerometers are perfect.
- Thrusters are ideal with the exception of saturation, that is, when thrusters reach their maximum available thurst force.
- Propulsion from main screws is ideal.
- Tracking of continuous trajectories is perfect.
- There are only small deviations from the desired posture, that is, the attitude and translational position.

The restrictions imposed are as follows:

- Hydrodynamic drag is neglected (due to low vehicle speed).

- Hydrodynamic added mass is limited to diagonal terms (actual added mass effects are not precisely known for the Phoenix AUV).

- There are no disturbances.

- The main screws operate at the same revolutions per minute and are treated as a single thruster.

- The vehicle has positive buoyancy.

The actual design of the hover mode autopilot controller is simplified with the design parameters defined above. The following chapter defines the model used and presents the development of the hovering autopilot.

D. SUMMARY

The preceding paragraphs present the problems addressed in the design and simulation of the NPS Phoenix AUV hover mode autopilot. The design parameters for which the autopilot is developed are presented. The defined design parameters consists of assumptions and restrictions which provide a means to bound the design efforts. This chapter includes a description of the problem from which the design efforts were focused. The next two chapters will present, in detail, the system design and simulation design, respectively.

IV. SYSTEM DESIGN

A. INTRODUCTION

This chapter will present in detail the design procedure followed in the development of the hover mode autopilot controller for the NPS Phoenix AUV. In designing the hover mode autopilot for the NPS Phoenix AUV, the vehicle dynamics are linearized in the earth coordinate system and then transformed to the body coordinate system. This choice was made because observation or interaction with an underwater object that uses a navigation system involves maneüvering in earth coordinates, not body coordinates.

In order to design the hover mode autopilot, the problem is divided into three parts. First, the equations of motion for each plane of a three-dimensional earth fixed coordinate system are determined. Second, linear control theory concepts are used to chose the control law equations and also to determine the gains required for feedback control. Finally, the necessary thruster forces in response to the autopilot are calculated using linear transformations.

The next chapter will discuss the simulation of the NPS Phoenix AUV in response to the autopilot. The following sections describe the design procedure and present the analytical steps.

B. EQUATIONS OF MOTION

The first section of this chapter discusses the equations of motion. Since the control design is linearized, we begin by dividing the three-dimensional space into three separate planes with the AUV's body coordinate system initially aligned with the fixed earth coordinate system. The planar views are arranged such that for the top view (Figure 1), the positive z axis is directed into the plane of the paper; the positive y axis comes out of the plane of the paper for the side view (Figure 2); and finally, for the stern view

(Figure 3), the positive x axis goes into the plane of the paper. Newton's second law, force equals mass times acceleration, was used to obtain the equations of motion. Note that the forces due to gravity and buoyancy are taken into account. Figure 1 through Figure 3 are referred to when deriving the equations of motion. In all three figures the origin also represents the center of gravity. For the NPS Phoenix AUV, the center of buoyancy (Cb) is the geometric center of the vehicle when submerged and is stationary. This is possible regardless of the posture because the hull cavity aspect does not change for a submerged vessel as opposed to a surface vessel whose partially submerged hull cavity physically changes with pitch and roll and because the NPS Phoenix AUV does not use variable ballast as is typical of manned submarines.

The equations of motion for the top view plane (Figure 1) can be obtained by using Newton's second law. Since the two axes that form the plane of Figure 1 are the x_E and y_E axis, the motions and forces concerned are described by these two coordinates. By studying Figure 1, we determine that the force in the x_E direction is equal to the mass times the acceleration in the x_E direction. We then rearrange the terms to solve for acceleration which yields the following equation:

$$\ddot{x}_E = \frac{F_{x_E}}{m} \qquad (1)$$

where $\ddot{x}_E$ denotes the acceleration along the x_E axis, m is the mass of the vehicle and F_{x_E} is the force applied to the vehicle along the x_E axis. A similar argument for the force in the y_E direction yields:

$$\ddot{y}_E = \frac{F_{y_E}}{m} \qquad (2)$$

where $\ddot{y}_E$ and F_{y_E} denotes the acceleration and force applied along the y_E axis respectively.

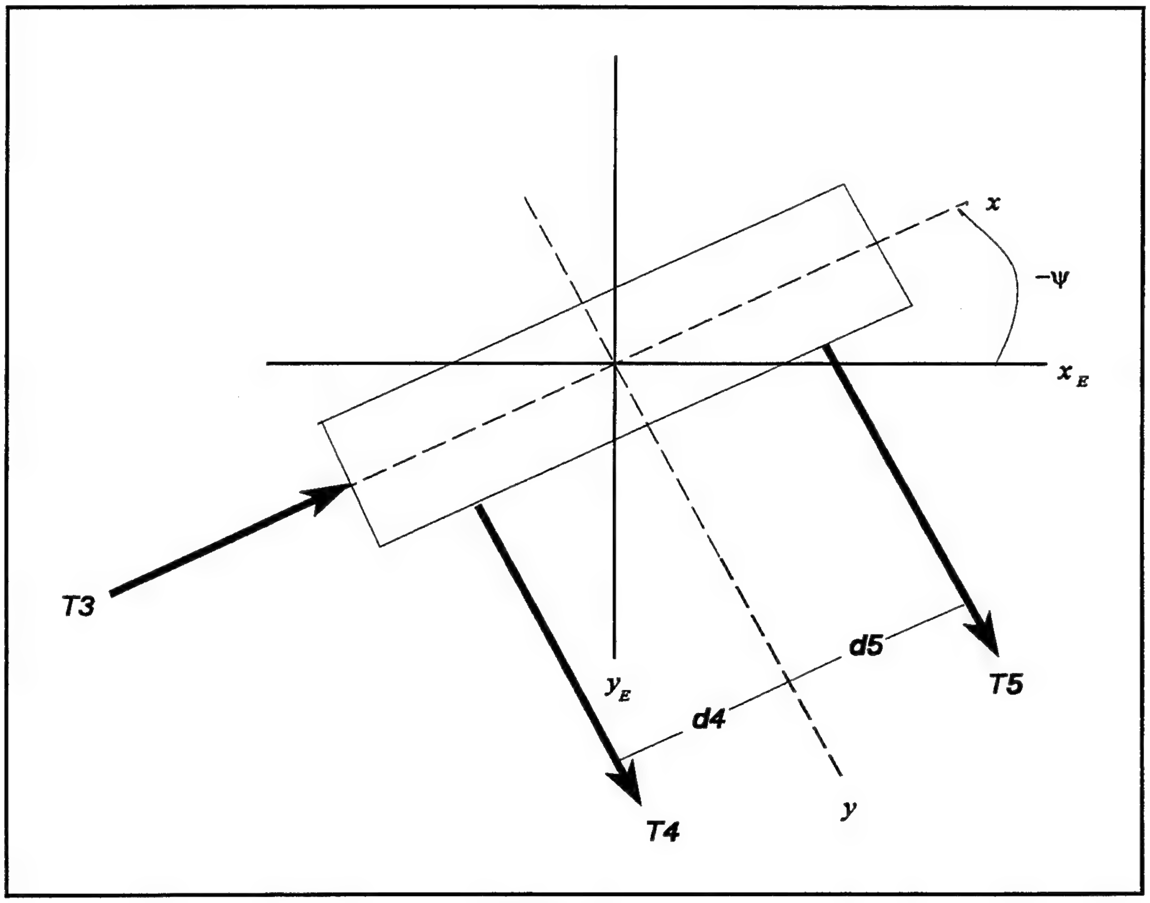

Figure 1

Top View of Yaw Motion Plane

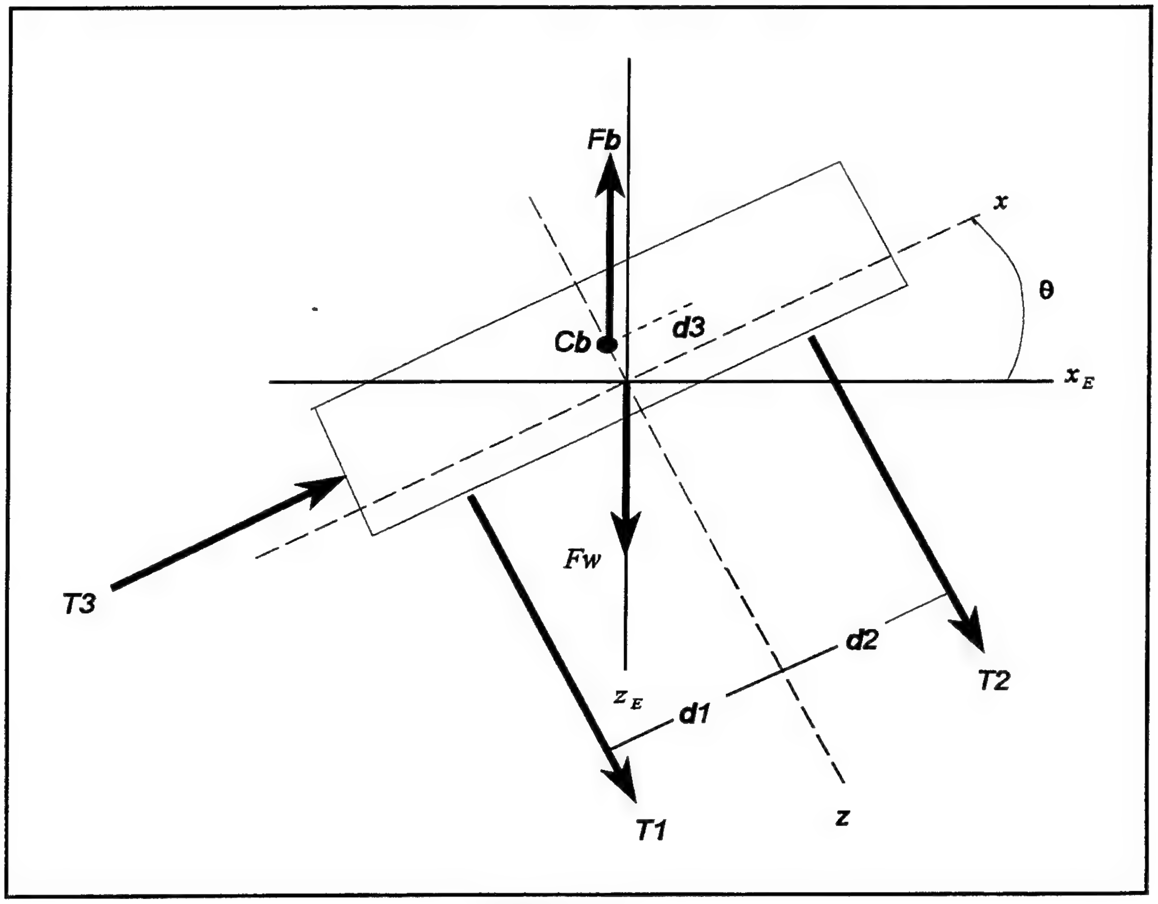

Figure 2

Side View of Pitch Motion Plane

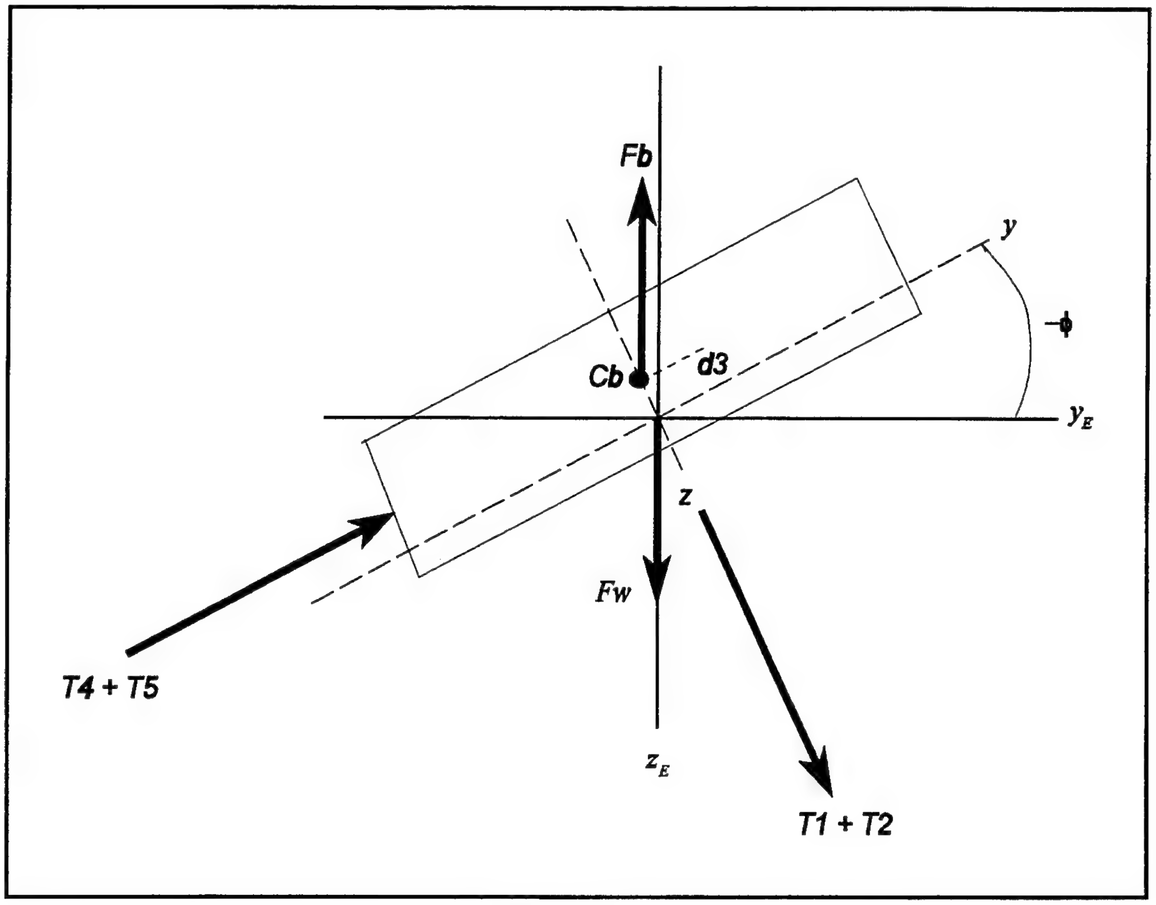

Figure 3

Stern View of Roll Motion Plane

There is one more force to contend with in this plane, and that is the torque about the z_E axis. As stated previously, the z_E axis is perpendicular to the x_E-y_E coordinate plane with positive z_E going into the plane of the paper. Here again Newton's second law can be applied to obtain a motion equation. By associating the force to torque, the mass to the moment of inertia and the angular acceleration to acceleration, we obtain Equation (3) from Figure 1:

$$\ddot{\Psi} = \frac{\Gamma_{\Psi}}{I_{zz}} \tag{3}$$

where $\ddot{\Psi}$ denotes the angular acceleration, Γ_{Ψ} is the torque about the z_E axis and I_{zz} is the inertia about the z axis. The total torque about the z_E axis is determined by the sum of thruster force times the distance of the respective thruster from the center of gravity. The forces and distances depicted in Figure 1 are used to determine Equation (4):

$$\Gamma_{\Psi} = -T4 \cdot d4 + T5 \cdot d5 \tag{4}$$

where T4d4 and T5d5 are the aft and forward lateral thrusters multiplied by their respective distances from the center of gravity.

For the side view (Figure 2), the two axes that form the coordinate plane of concern are the x_E and z_E axes. Since the x_E of the side view coincides with that of the top view, the equation of motion is the same; thus, Equation (1) is used for both the x_E-z_E and the x_E-y_E planes. When dealing with the z_E axis, there are two additional forces to include, buoyancy and weight, which are always applied at the center of buoyancy and the center of gravity, respectively. The vehicle is operated with positive buoyancy so that in the event of a system failure that causes loss of power, the vehicle will float to the surface for recovery. The equation of motion in the z_E direction derived from Figure 2 is :

$$\ddot{z}_E = \frac{Fz_E}{m} + \frac{Fw}{m} - \frac{Fb}{m} \qquad (5)$$

As in the x_E-y_E coordinate plane, there is also a force due to a torque involved; however, in this case the torque is about the y_E axis. By observing Figure 2 and following the same procedure as before, we get Equation (6):

$$\ddot{\theta} = \frac{\Gamma_\theta}{I_{yy}} \qquad (6)$$

where $\ddot{\theta}$ denotes the angular acceleration about the y_E axis, Γ_θ represents the torque about the y_E axis, and I_{yy} is the inertia. In order to determine the total torque from Figure 2, a more involved analysis is required due to the righting moment effect caused by the buoyancy (Fb) and gravitational forces (Fw). By applying vector analysis rules, we can obtain the following torque due to the righting moment:

$$\Gamma_{RM} = Fb \cdot d3 \cdot \sin\theta \qquad (7)$$

The total torque about the y_E axis is then determined by summing all the torques about the y_E axis, thus yielding Equation (8).

$$\Gamma_\theta = T1 \cdot d1 - T2 \cdot d2 - T3 \cdot d3 - \Gamma_{RM} \qquad (8)$$

The axes for the stern view (Figure 3) are y_E and z_E. The equations of motion for these two axes have already been determined from the previous two coordinate planes discussed. The remaining force in the y_E-z_E coordinate plane is the torque about the x_E axis:

$$\ddot{\phi} = \frac{\Gamma_\phi}{I_{xx}} \qquad (9)$$

where $\ddot{\phi}$ denotes the angular acceleration about the x_E axis, Γ_ϕ represents the torque about the x_E axis, and I_{xx} is the inertia. In this case there is no active means to control roll; therefore, the hydrodynamic damping effect is used to counter the torque induced by the lateral thrusters and the righting moment due to the rolling effect. Thus, the equation for torque about the x_E axis is

$$\Gamma_\phi = T4 \cdot d3 + T5 \cdot d3 - Fb \cdot d3 \cdot \sin\phi - k_\phi \cdot p \tag{10}$$

All the equations of motion for the three planes have been derived. In order to use these equations, some assumptions and substitutions will be incorporated into them. First, since the vehicle will be operating completely submerged there is an added mass effect due to the water that will be moving with the AUV. These terms are identified as surge, sway, heave, roll, pitch and yaw hydrodynamic coefficients and have been approximated in (Warner, 1991). However, only the diagonal terms ($x_{\dot{u}}, y_{\dot{v}}, z_{\dot{w}}, K_{\dot{p}}, M_{\dot{q}}, N_{\dot{r}}$) will be used as mentioned in Chapter III under the limitations imposed. Hence, the mass and inertias are divided into three components each where the mass equations become

$$m_x = m + x_{\dot{u}} \tag{10a}$$

$$m_y = m + y_{\dot{v}} \tag{10b}$$

$$m_z = m + z_{\dot{w}} \tag{10c}$$

and the moments of inertia equations become

$$I_{xx} = I_{xx} + K_{\dot{p}} \tag{10d}$$

$$I_{yy} = I_{yy} + M_{\dot{q}} \tag{10e}$$

$$I_{zz} = I_{zz} + N_{\dot{r}} \tag{10f}$$

The average of m_x, m_y, and m_z, denoted as $\bar{m}$, is substituted into Equations (1), (2) and (5) to yield Equations (11), (12) and (13), respectively. For the moments of inertia a similar approach is taken except that since I_{xx} is small compared to I_{yy} and I_{zz} it is neglected. The average of I_{yy} and I_{zz} is used and denoted as I which is substituted into Equations (3), (6), and (9).

The next consideration is the torques introduced by the thruster commands. Note that for Γ_{Ψ} there are no nonlinear terms induced by the horizontal thrusters T4 and T5; hence, the resulting torque is the torque commanded and is denoted as $\Gamma_{\Psi com}$. $\Gamma_{\Psi com}$ is substituted into Equations (3) and (4) to yield Equation (14). There is, however, a nonlinear term involved for Γ_{θ}. The nonlinear term here is a result of the righting moment due to the buoyancy force, the pitch angle, and the distance between the center of buoyancy and the center of gravity. Therefore, the torque commanded, $\Gamma_{\theta com}$, is only part of the total torque. The following section discusses how this nonlinearity is dealt with. For now let the commanded torque be defined as

$$\Gamma_{\theta com} = T1d1 - T2d2 - T3d3 \tag{10g}$$

and substitute it into Equations (6) and (8) to obtain Equation (15). Since roll is not actively controlled, no attempt was made to determine a command torque about the x axis. A summary of the motion equations with the torque equations substituted into their respective motion equations is provided in Table 1. Note that the equations have been solved for the forces and torques and no attempt has been made to derive a torque commanded equation for the moment about the x axis. The following section discusses the control theory involved for determining the control law equations.

$F_{x_E} = \ddot{x}_E \bar{m}$	(11)	$\Gamma_{\Psi com} = \ddot{\Psi} \bar{I} = -T4d4 + T5d5$	(14)
$F_{y_E} = \ddot{y}_E \bar{m}$	(12)	$\Gamma_{\theta com} = \ddot{\theta} \bar{I} = T1d1 - T2d2 - T3d3 - Fbd3 \sin \theta$	(15)
$F_{z_E} = \ddot{z}_E \bar{m} + Fb - Fw$	(13)	$\Gamma_{\phi} = \ddot{\phi} I_x = T4d3 + T5d3 - Fbd3 \sin \phi - k_{\phi} p$	(16)

Table 1

Equations of Motion

C. CONTROL THEORY

Based on the design environment defined in the previous chapter and the model defined in the previous section, linear control theory is selected for the development of the hover mode autopilot controller. The following paragraphs address the linear control theory concepts used in the design of the autopilot.

The control law equations were selected by using a control law of form

$$f(t) = - k_1 x_1(t) - k_2 x_2(t) + b(t) \tag{17}$$

where f(t) is the resulting force in earth coordinates, k_1 and k_2 are the gains, $x_1(t)$ and $x_2(t)$ are position and velocity respectively, and b(t) is a biasing term. The biasing term is used to cancel out any nonlinear terms resulting from the righting moments due to interaction of buoyancy and gravity. Cancelling out the nonlinear terms with the biasing term linearizes the system.

In order to determine the gains to be used for the design, the damping ratio (ζ) and the natural frequency (ω_n) must be known. The system is designed to be slightly underdamped, hence 0.8 was selected for the damping ratio. A settling time of 125 seconds was chosen for each plane based on an assumption that it was a reasonable amount of time for the vehicle to move fifteen feet forward, ten feet laterally and five feet in depth. The settling time equation used from linear control theory is

$$t_s = 4/\zeta\omega_n \tag{18}$$

This equation is based on a tolerance of a two percent range of the desired final value (Thompson, 1989). With Equation (18), given the settling time and damping ratio, the natural frequency is calculated. The initial conditions are set to zero and the control law equations are substituted into the equations of motion to obtain the characteristic equations which are of the form

$$\lambda^2 + 2\zeta\omega_n\lambda + \omega_n = 0 \tag{19}$$

The required thruster forces are then obtained by linear transformation of the forces in earth coordinates. The forces in body coordinates are calculated by extracting the direction cosine transformation matrix from the homogeneous transformation matrix, discussed in the next chapter, and pre-multiplying the result with the forces in earth coordinates which will be determined from the control law equations. The inverse of the thruster coefficient matrix is then pre-multiplied with the forces in body coordinates to obtain the necessary thruster forces in response to the hover mode autopilot. A more detailed analysis of the preceding discussion is presented in the following sections.

D. CONTROL EQUATIONS

In order to simplify the design of the hover mode autopilot, the system is linearized. This is accomplished by adding or subtracting biasing terms where nonlinearities exist. A linear feedback control law of the form as Equation (17) is chosen for each of the forces and torques presented in Table 1 except for Equation (16) which is not actively controlled. Additionally, note that the torque about the z_E axis has no nonlinear terms which allows it to be controlled by using the lateral thrusters T4 and T5. For the torque about the y_E axis there does exist a nonlinear term which is the result of the righting moment effect due to buoyancy and gravity. As discussed earlier, this is resolved by adding a biasing term to the control. Also note that constants are included for the control equations in the z_E direction to compensate for the effects of buoyancy and gravity. The resulting equations are as follows:

$$F_{x_E} = -k_x(x_E - x_o) - k_{\dot{x}}\,\dot{x}_E \tag{20}$$

$$F_{y_E} = -k_y(y_E - y_o) - k_{\dot{y}}\,\dot{y}_E \tag{21}$$

$$F_{z_E} = -k_z(z_E - z_o) - k_{\dot{z}}\,\dot{z}_E + Fb - Fw \tag{22}$$

$$\Gamma_{\Psi com} = -k_{\Psi}(\Psi - \Psi_o) - k_{\dot{\Psi}}\,\dot{\Psi} \tag{23}$$

$$\Gamma_{\theta com} = -k_{\theta}(\theta - \theta_o) - k_{\dot{\theta}}\,\dot{\theta} + Fb \cdot d3 \cdot \sin\theta \tag{24}$$

Now that the control equations have been determined, the necessary gains can be calculated. Substituting Equations (11) - (13) into Equations (20) - (24) and solving for zero, one obtains the following:

$$\ddot{x}_E + \frac{k_{\dot{x}}}{m}\dot{x}_E + \frac{k_x}{m}(x_E - x_o) = 0 \tag{25}$$

$$\ddot{y}_E + \frac{k_{\dot{y}}}{m}\dot{y}_E + \frac{k_y}{m}(y_E - y_o) = 0 \tag{26}$$

$$\ddot{z}_E + \frac{k_{\dot{z}}}{m}\dot{z}_E + \frac{k_z}{m}(z_E - z_o) = 0 \tag{27}$$

$$\ddot{\theta} + \frac{k_{\dot{\theta}}}{I}\dot{\theta} + \frac{k_{\theta}}{I}(\theta - \theta_o) = 0 \tag{28}$$

$$\ddot{\Psi} + \frac{k_{\dot{\Psi}}}{I}\dot{\Psi} + \frac{k_{\Psi}}{I}(\Psi - \Psi_o) = 0 \tag{29}$$

which are of the form as Equation (19) presented earlier. If one sets the initial conditions to zero and associates like terms to that of Equation (19), the calculation of the various gains is made possible. Since the settling time was chosen to be the same for each control equation, all like terms for the gains were set equal to each other. Given a settling time of 125 seconds and a damping ratio of 0.8, Equation (18) is solved for the natural frequency which results as .04 radians per second. The gains are determined by using the following formulas:

$$k_x = k_y = k_z = \omega_n^2 \bar{m} \tag{29a}$$

$$k_{\dot{x}} = k_{\dot{y}} = k_{\dot{z}} = 2\zeta\omega_n \bar{m} \tag{29b}$$

$$k_\Psi = k_\theta = \omega_n^2 I \tag{29c}$$

$$k_{\dot{\Psi}} = k_{\dot{\theta}} = 2\zeta\omega_n I \tag{29d}$$

for the forces and torques respectively. All that remains for the design of the hover mode autopilot is to use the previously derived control equations and gains to determine the required thruster forces, which is discussed in the next section.

E. THRUSTER EQUATIONS

The thruster forces in response to the derived control equations are determined in this section. This is done by studying the thruster vectors depicted in Figures 1-3.

The forces acting on the vehicle in body coordinates are observed to be those resulting from the thrusters; hence, the resulting body forces are

$$F_x = T3 \tag{30}$$

$$F_y = T4 + T5 \tag{31}$$

$$F_z = T1 + T2 \tag{32}$$

or more conveniently,

$$F = \begin{bmatrix} T3 \\ T4+T5 \\ T1+T2 \end{bmatrix} \tag{33}$$

In order to determine the forces in body coordinates, the forces in earth coordinates must be transformed into body forces. The convention most commonly used to accomplish this is by means of the direction cosine transformation matrix discussed in (Cooke, 1992) which defines the direction cosine transformation matrix as

$$R = \begin{bmatrix} \cos\psi\cos\theta & \cos\psi\sin\theta\sin\phi-\sin\psi\cos\phi & \cos\psi\sin\theta\cos\phi+\sin\psi\sin\phi \\ \sin\psi\cos\theta & \cos\psi\cos\phi+\sin\psi\sin\theta\sin\phi & -\cos\psi\sin\phi+\sin\psi\sin\theta\cos\phi \\ -\sin\theta & \cos\theta\sin\phi & \cos\theta\cos\phi \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \tag{34}$$

The transformation from body to earth coordinates is given by

$$\mathbf{F_E} = \mathbf{R\,F} \tag{34a}$$

Thus, using matrix algebra yields a transformation from earth to body coordinates as

$$\mathbf{F} = \mathbf{R}^{-1}\,\mathbf{F_E} \tag{34b}$$

Note that the direction cosine matrix has a unique characteristic in that its inverse is equal to its transpose, thus $\mathbf{R}^{-1} = \mathbf{R}^{T}$ and $\mathbf{F} = \mathbf{R}^{-1}\,\mathbf{F_E}$ can be rewritten as $\mathbf{F} = \mathbf{R}^{T}\,\mathbf{F_E}$. By substituting Equations (30) - (32) for **F** and Equations (20) - (22) for $\mathbf{F_E}$, we obtain a set of three equations and five unknowns, namely the thruster forces T1 through T5. The two torque equations must be used obtain a set of five equations and five unknowns to be able to solve for the thruster forces. By combining Equations (14) and (15) with Equations (23) and (24) , respectively, we get

$$-T4 \cdot d4 + T5 \cdot d5 = \Gamma_{\Psi com} \tag{35}$$

$$T1 \cdot d1 - T2 \cdot d2 - T3 \cdot d3 = \Gamma_{\theta com} \tag{36}$$

If we group Equations (30) - (32) and Equations (35) - (36) into a matrix form and solve for the thruster vector, the following result is obtained

$$\begin{bmatrix} T1 \\ T2 \\ T3 \\ T4 \\ T5 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -d4 & d5 \\ d1 & -d2 & -d3 & 0 & 0 \end{bmatrix}^{-1} \begin{bmatrix} F_x \\ F_y \\ F_z \\ \Gamma_{\Psi com} \\ \Gamma_{\theta com} \end{bmatrix} \tag{37}$$

To determine the Euler angles and Euler angle rates needed in the torque equations, the rotational velocities of the vehicle, measured about the vehicles body coordinates, are transformed to Euler angle rates using the non-orthogonal transformation matrix defined in (Frank, 1969). These rates are then integrated to obtain the Euler angles. For the simulation of the hover mode autopilot, the Newton-Euler equations are used to provide the body translational and rotational accelerations. By integrating these accelerations, we get the body translational and rotational velocities. This process will be discussed in further detail in the next chapter.

The horizontal and vertical thrusters have a maximum thrust force of approximately two pounds. The combined thrust force of the main screws is approximately ten pounds. These limits have been implemented into the simulation model. Thus, if a desired posture requires more force than is available, the thrusters will saturate.

F. SUMMARY

This chapter presents a detailed analysis of the procedure followed in the design of the hover mode autopilot for the NPS Phoenix AUV. A simplified mathematical model is the basis for the initial autopilot design from which a more complex model may be derived in future work. Control theory provides the essential tools for which the hover mode autopilot controller has been designed. Although the actual system is nonlinear, the use of biasing terms allows the system to be linearized which permits the use of linear control theory concepts in the design of the autopilot, thus reducing the complexity of the design. The equations of motion were determined, followed by a brief section on control theory. Next the derivation of the control equations were determined, concluding with the development of the thruster equations. The results obtained from the testing and simulation satisfy the requirements and are discussed in Chapter VI. Chapter V presents the simulator design.

V. SIMULATOR DESIGN

A. INTRODUCTION

The simulation of the hover mode autopilot for this thesis takes on two phases, numerical simulation and visual simulation. The numerical simulation phase produces the values as calculated from mathematical models which are plotted versus time such that they may be analyzed. Visual simulation takes the values produced in the numerical phase and applies them to a visual simulator which presents a pictorial observation of how the vehicle is responding to the autopilot. These two phases are performed simultaneously. The following sections will present the procedure for simulator design. It begins with the dynamic equations, followed by the various transformations, and the method used to display the vehicle movement, concluding with the force and moment calculations.

B. DYNAMIC EQUATIONS

In order to properly simulate any system, a mathematical model must be developed which closely represents the system. The previous chapter derived the various mathematical models which represent the type of control to be used. Here the mathematical model for the vehicle dynamics will be illustrated.

Before continuing, it is imperative to define the meaning of the terms that will be used. Table 2 accomplishes this task in a concise manner. The terms defined in Table 2 and Figures 4 - 6 are used to derive the dynamic equations. Since the translational and rotational accelerations in the body reference frame are the most convenient to measure, it makes sense to solve for the accelerations. Figure 4 depicts a body coordinate system with the various velocities necessary to solve for the translational acceleration in the x direction without the effect of gravity.

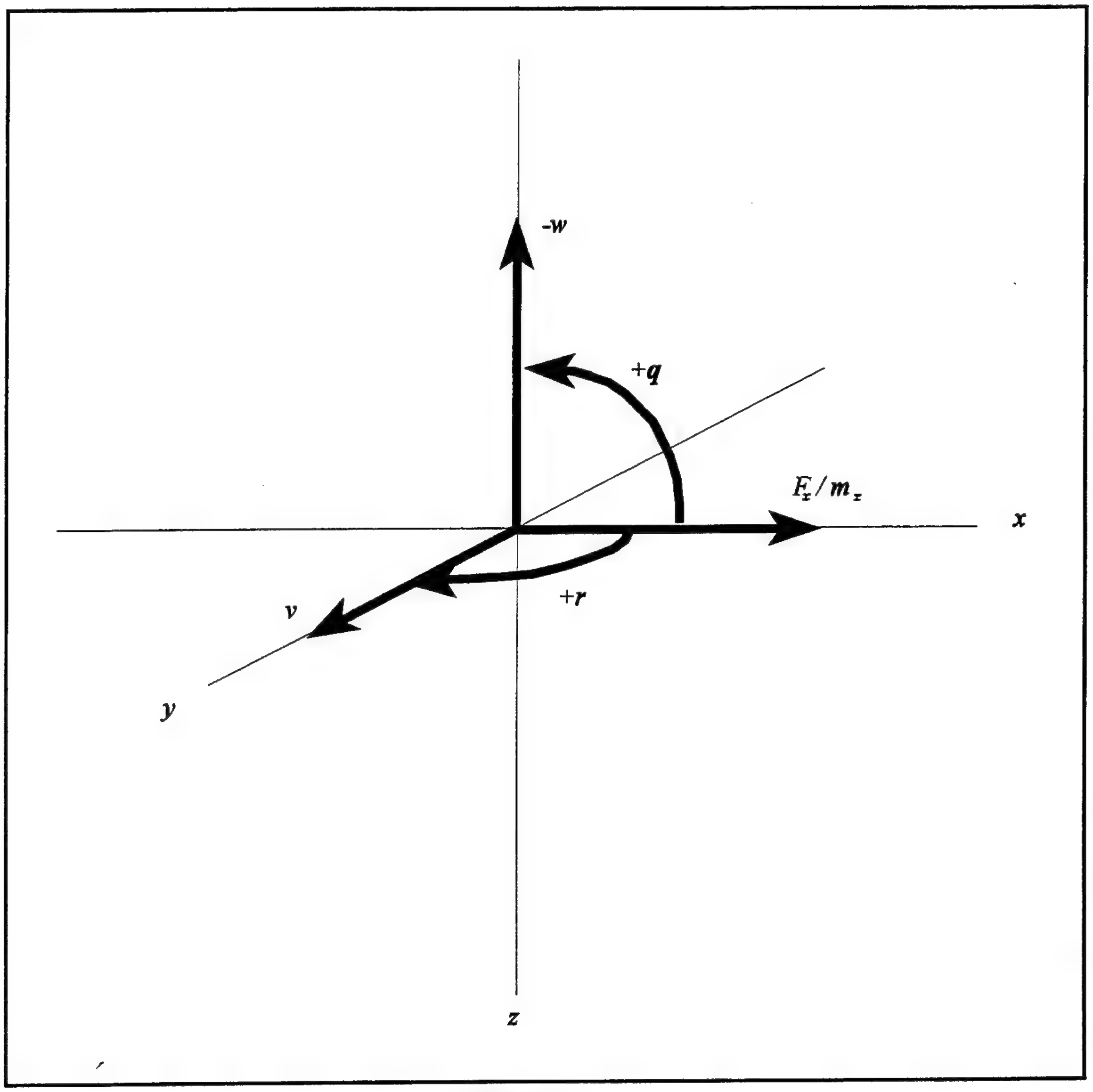

Figure 4

Longitudinal Velocity Growth Rate Derivation

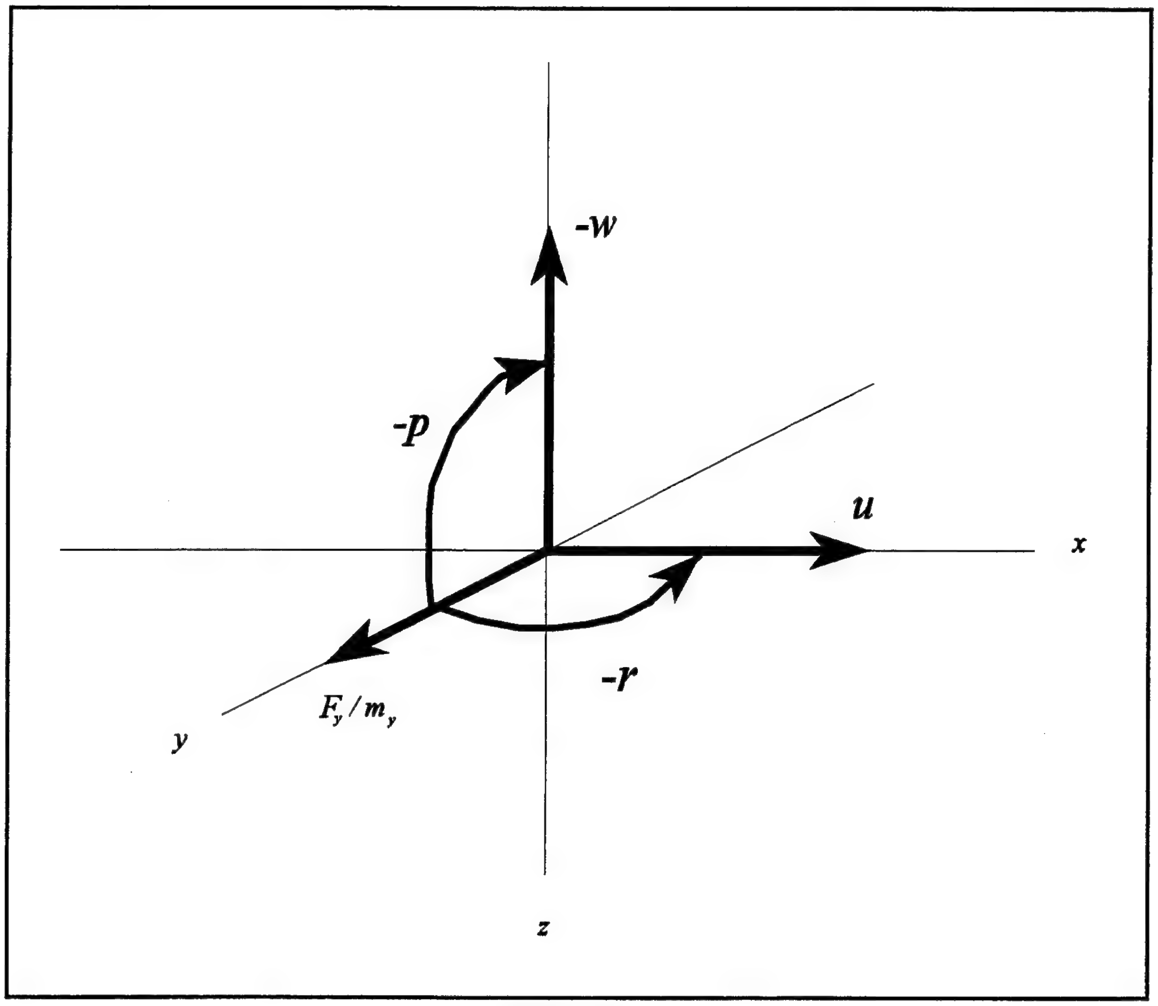

Figure 5

Lateral Velocity Growth Rate Derivation

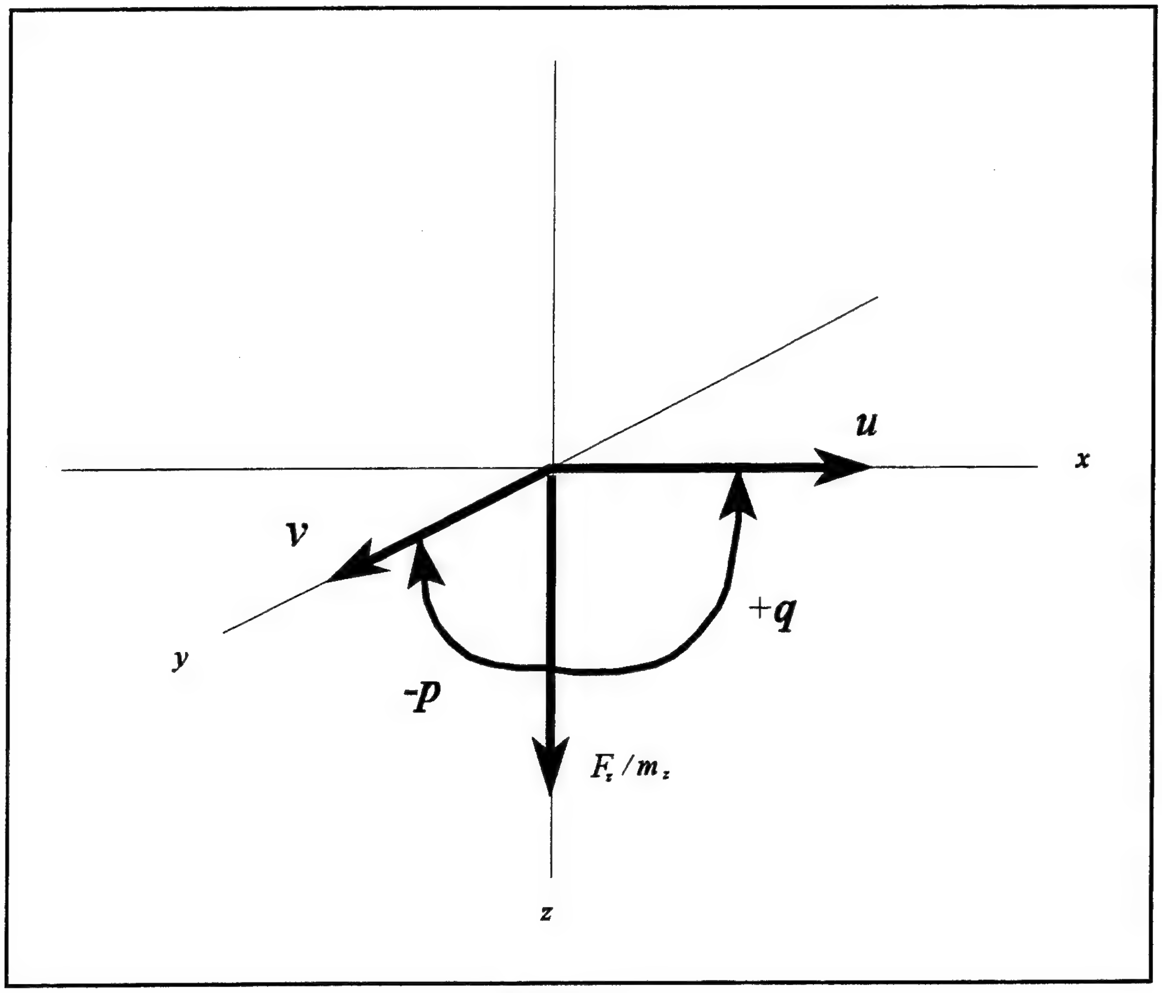

Figure 6

Vertical Velocity Growth Rate Derivation

Symbol	Definition	Units
x, y, z	longitudinal, lateral, and vertical body coordinate axes respectively	ft
u, v,w	x, y, z translational velocities	ft/sec
$\dot{u},\dot{v},\dot{w}$	x, y, z translational velocity growth rates	ft/sec^2
ϕ, θ, Ψ	roll, elevation, and azimuth in earth coordinates	rad
p, q, r	rotational velocities about the x, y, and z axes respectively	rad/sec
$\dot{p},\dot{q},\dot{r}$	rotational velocity growth rates about the x, y, and z axes respectively	rad/sec^2
F_x,F_y, F_z	forces acting on the body in the x, y, and z direction	lbs
K, M, N	torques acting on the body about the x, y and z axes respectively[1]	ft lbs
I_{xx}, I_{yy}, I_{zz}	moments of inertia of the body about the x, y, and z axes respectively	slugs ft^2

Table 2

Conventional Variable Definitions

The total translational acceleration in the x direction is the sum of all the x component acceleration vectors. For this case, there are three such components. The first term is due to the effects of the translational velocity in the y direction and the rotational velocity about the z axis. Note that multiplying a translational velocity by a rotational velocity will yield the tangential component of acceleration which is mutually perpendicular to both the translational velocity vector and the rotational velocity. Similarly, the second term is due to the effects of the translational velocity in the z direction and the rotational velocity about the y axis. The third acceleration component in the x direction is due to the force being applied in the x direction.

There is actually one more acceleration vector that affects the body and that is the acceleration due to gravity. Since gravity is a constant that is always present, it will

[1] Although the equations for these values are the same for aircraft and ships, the convention for notation is different. When dealing with aircraft, the torque about the x axis is denoted by L; but when dealing with ships, L is reserved for length and so K is used instead.

always affect the body. Gravity is an acceleration in earth coordinates and must be transformed into body coordinates. Let the vector **A** be defined as $[0\ 0\ g]^T$, the constant accelerations in earth coordinates. The acceleration vectors in body coordinates are produced by $\mathbf{R}^{-1}\mathbf{A}$ which yields the following:

$$a_x = -g \sin\theta \tag{38}$$

$$a_y = g \cos\theta \sin\phi \tag{39}$$

$$a_z = g \cos\theta \cos\phi \tag{40}$$

where a_x, a_y, and a_z are the accelerations due to gravity in the x, y, and z directions respectively. Thus, the following equation is obtained from studying Figure 4 and includes Equation (38) of the previous analysis on the effects of gravity.

$$\dot{u} = vr - wq + \frac{F_x}{m_x} - g \sin\theta \tag{41}$$

Studying Figure 5, we see that the acceleration in the y direction is the sum of the translational velocity (p) times the rotational velocity (w), the translational velocity (u) times the rotational velocity (r), and the acceleration due to force in the y direction. The acceleration in the y direction due to gravity, Equation (39), is added and results in Equation (42). The same procedure is followed, this time using Figure 6 and including Equation (40) which yields Equation (43).

$$\dot{v} = wp - ur + \frac{F_y}{m_y} + g \cos\theta \sin\phi \tag{42}$$

$$\dot{w} = uq - vp + \frac{F_z}{m_z} + g \cos\theta \cos\phi \tag{43}$$

The angular accelerations are obtained by analyzing the torques involved. First begin with the angular acceleration about the x axis. The torque about the x axis is equal to the rotational velocity growth rate about the x axis ($\dot{p}$) times the moment of inertia (I_{xx}), which is equal to the sum of its individual components. One of the components (K) is merely the resultant torque of the applied forces. The next component is determined by taking the angular momentum about the y axis, that is the product of the moment of inertia about the y axis (I_{yy}) and the rotational velocity about the y axis (q), and multiplying it with the rotational velocity about the z axis (r). Finally, the third component, is obtained by taking the angular momentum about the z axis, I_{zz} times r, and multiplying it with the rotational velocity about the y axis (-q). Since the rotational velocity growth rate $\dot{p}$ is the term desired, both sides of the equation are divided by I_{xx}, ultimately resulting in Equation (44). A similar analysis in determining $\dot{q}$ and $\dot{r}$ yields Equations (45) and (46).

$$\dot{p} = \frac{[\,(I_{yy}-I_{zz})\,qr + K\,]}{I_{xx}} \tag{44}$$

$$\dot{q} = \frac{[\,(I_{zz}-I_{xx})\,rp + M\,]}{I_{yy}} \tag{45}$$

$$\dot{r} = \frac{[\,(I_{xx}-I_{yy})\,pq + N\,]}{I_{zz}} \tag{46}$$

Equations (41) to (46) constitute the Newton-Euler equations used as the mathematical model to represent the vehicle dynamics. These equations are implemented in a function named "update-velocity-growth-rate". We integrate Equations (41) - (46) by using the Euler integration method in a function called "update-velocity". This function will produce the translational and rotational velocities in body coordinates which will be used to determine the vehicle position in earth coordinates. The functions just mentioned as well as all the rest of the LISP source code will be discussed in Section D. This will be done by coordinate transformations which are discussed in the next section. The results are also used to calculate the required thruster forces necessary to maneuver the vehicle to the desired posture.

C. TRANSFORMATIONS

To describe three-dimensional objects, a three dimensional coordinate system is needed. A three-dimensional Cartesian system is commonly used due to the ease at which it can be manipulated using matrix algebra. For this thesis the object is constructed with polygons (Figure 7).

This section discusses the use of the Homogeneous Transformations, Euler Angle Transformations and Non-orthogonal transformations for the three dimensional simulation of the NPS Phoenix AUV in response to the hover mode autopilot. The purpose, application, and implementation of each are discussed beginning with the Homogeneous transformation.

1. Homogeneous Transformation

The Homogeneous Transformation is a homogeneous coordinate system which includes both rotation and translation values (Paul, 1981). Rotation refers to pivoting the object about its axes where azimuth is the angle about the z axis, elevation is the angle about the rotated y axis and roll is the angle about the rotated x axis. Translation refers to moving an object to a new point in the coordinate system while maintaining its orientation. A Homogeneous Transformation is the result of a rotation followed by a translation. It is this transformation that represents general three-dimensional movement of a rigid body figure.

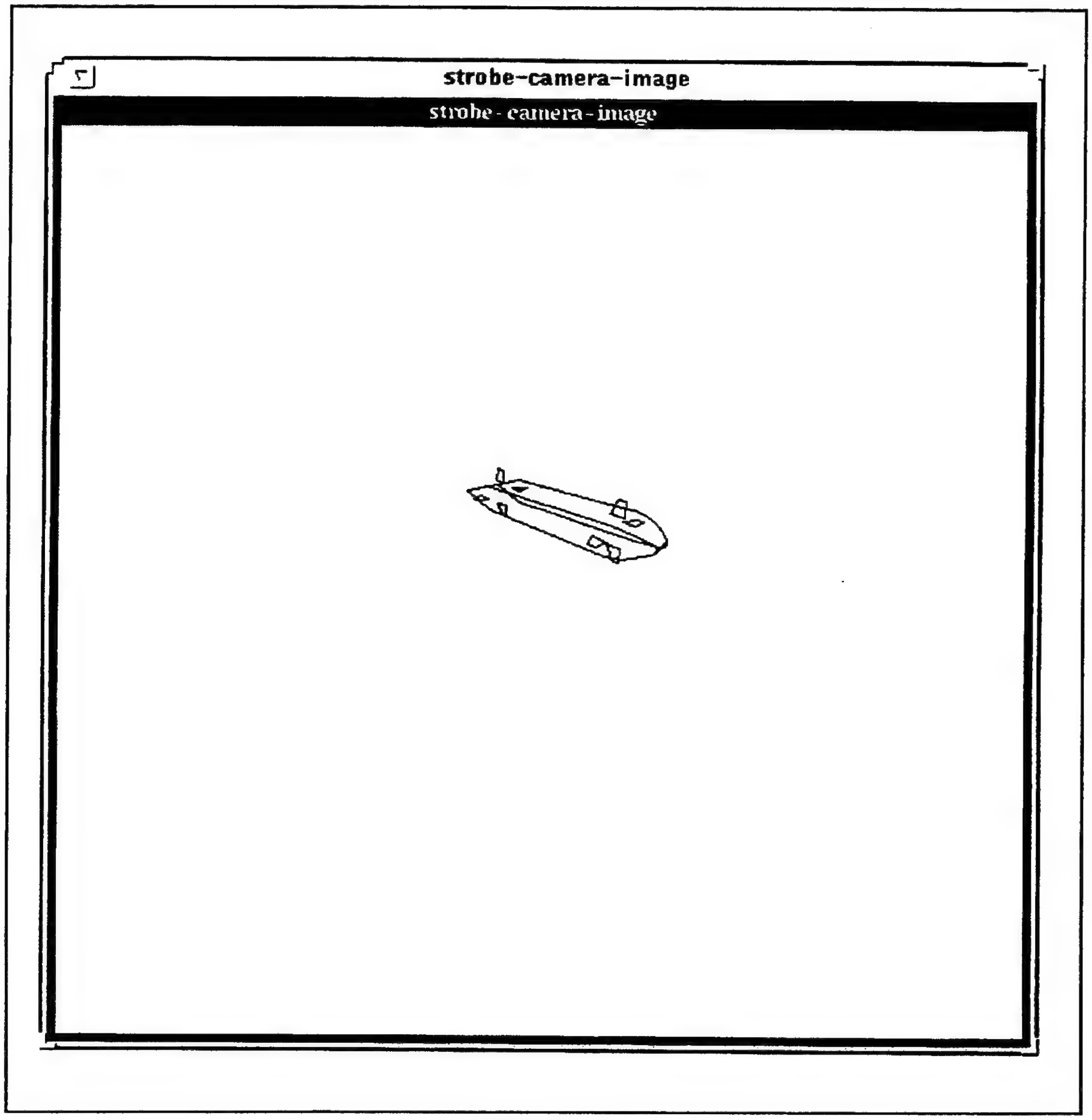

Figure 7

Polygon Representation of NPS Phoenix AUV

$$H = \begin{bmatrix} \cos\psi\cos\theta & \cos\psi\sin\theta\sin\phi - \sin\psi\cos\phi & \cos\psi\sin\theta\cos\phi + \sin\psi\sin\phi & dx \\ \sin\psi\cos\theta & \cos\psi\cos\phi + \sin\psi\sin\theta\sin\phi & -\cos\psi\sin\phi + \sin\psi\sin\theta\cos\phi & dy \\ -\sin\theta & \cos\theta\sin\phi & \cos\theta\cos\phi & dz \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & dx \\ r_{21} & r_{22} & r_{23} & dy \\ r_{31} & r_{32} & r_{33} & dz \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (47)$$

The Homogeneous transformation matrix is (McGhee, 1994) defined by the above equation. Note that Equation (47) is merely the three by three direction cosine matrix expanded to a four by four matrix where the last column represents the amount of translation. This transformation accomplishes a rotation followed by a translation. The LISP function called "homogeneous-transform" located in the file named "robot-kinematics.cl" performs the action for the simulation. Equation (47) is used to rotate and translate each node, which is done by a LISP method called "transform-node-list" in a file named "update-functions.cl". A node is a vector consisting of the x, y, and z coordinates of the three-dimensional coordinate system augmented with a value of one as the fourth coordinate. Performing the above actions on each node of the polygon figure produces the three dimensional effect of motion. The values used in the homogeneous transformation are in earth coordinates, which are obtained from converting the body velocities into earth velocities and then integrating them. Further details of this process follow in the next two sub-sections.

2. Euler Angle Transformation

Euler Angle Transformations are used to transform vectors between the earth coordinate system and the body coordinate system. This transformation is used three times in the simulation. The first time it is used in conjunction with the Homogeneous Transformation, since it is the result of the rotation about the axes. The second time it is used, its inverse is taken and pre-multiplied with the body translational velocities to obtain the translational rates in earth coordinates for use in the control equations. Finally, the Euler Angle Transformation matrix is used to calculate the forces in body coordinates when solving for the thruster forces.

In order to simulate the dynamics of the AUV, the Newton-Euler equations are used as the mathematical model to obtain the translational and rotational body accelerations which are integrated to get the translational and rotational body velocities. In actuality, the acceleration and velocity values would be generated by the inertial navigation system. In either case, the body velocities must be transformed into earth velocities. By integrating the earth velocities, we get the position in earth coordinates. The translational velocities, or any vector for that matter, can be converted between body coordinates and earth coordinates using the direction cosine matrix as a function of Euler angles; this matrix has been defined by Equation (34). Thus, the relationship **E=RB**, where **E** represents any three by one earth vector, **R** is the direction cosine matrix previously defined, and **B** is any three by one body vector, is used to convert the translational body velocities into earth velocities (Frank, 1969) resulting in the following,

$$\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{z}\end{bmatrix} = \begin{bmatrix}\cos\psi\cos\theta & \cos\psi\sin\theta\sin\phi-\sin\psi\cos\phi & \cos\psi\sin\theta\cos\phi+\sin\psi\sin\phi\\ \sin\psi\cos\theta & \cos\psi\cos\phi+\sin\psi\sin\theta\sin\phi & -\cos\psi\sin\phi+\sin\psi\sin\theta\cos\phi\\ -\sin\theta & \cos\theta\sin\phi & \cos\theta\cos\phi\end{bmatrix}\begin{bmatrix}u\\ v\\ w\end{bmatrix} \qquad (48)$$

Since the direction cosine matrix is computed as part of the homogeneous transformation matrix, it is extracted via a LISP method called "extract-R-matrix". This action avoids additional computations when determining the **R** matrix. The above procedure has only transformed the translational velocities, the next subsection describes how the rotational velocities are transformed.

3. Non-orthogonal Transformation

The Non-orthogonal transformation is used to transform the body rotational velocities to Euler angle rates. This is necessary for numerical integration of Euler rates to obtain Euler angles.

The rotational body accelerations that are obtained from the Newton-Euler equations are integrated yielding the rotational body velocities. Since these vectors are not orthogonal, a non-orthogonal transformation is required. (Frank, 1969) defines the non-orthogonal transformation matrix as

$$T = \begin{bmatrix} 0 & \cos\phi & -\sin\phi \\ 1 & \tan\theta\sin\phi & \tan\theta\cos\phi \\ 0 & \sec\theta\sin\phi & \sec\theta\cos\phi \end{bmatrix} \tag{49}$$

Equation (49) provides the means for transforming the rotational body velocities to earth velocities by pre-multiplying the vector $[p\ q\ r]^T$ with the non-orthogonal transformation matrix as shown below.

$$\begin{bmatrix} \dot{\theta} \\ \dot{\phi} \\ \dot{\Psi} \end{bmatrix} = \begin{bmatrix} 0 & \cos\phi & -\sin\phi \\ 1 & \tan\theta\sin\phi & \tan\theta\cos\phi \\ 0 & \sec\theta\sin\phi & \sec\theta\cos\phi \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{50}$$

The resultant vector from Equation (50) is integrated to give us the Euler Angles θ, ϕ, and Ψ. These angles are used in the Homogeneous Transformation matrix as well as the Euler Angle Transformation matrix. They are also important in determining the required control torques.

D. LISP CODE

Appendix A contains the entire LISP source code for this thesis. This section serves to describe the purpose of the essential functions utilized in the implementation and simulation of the hover mode autopilot. Due to the complexity of the code, the simulation cycle is best described using a flowchart (Figure 8). Refer to Table 2 for the symbol

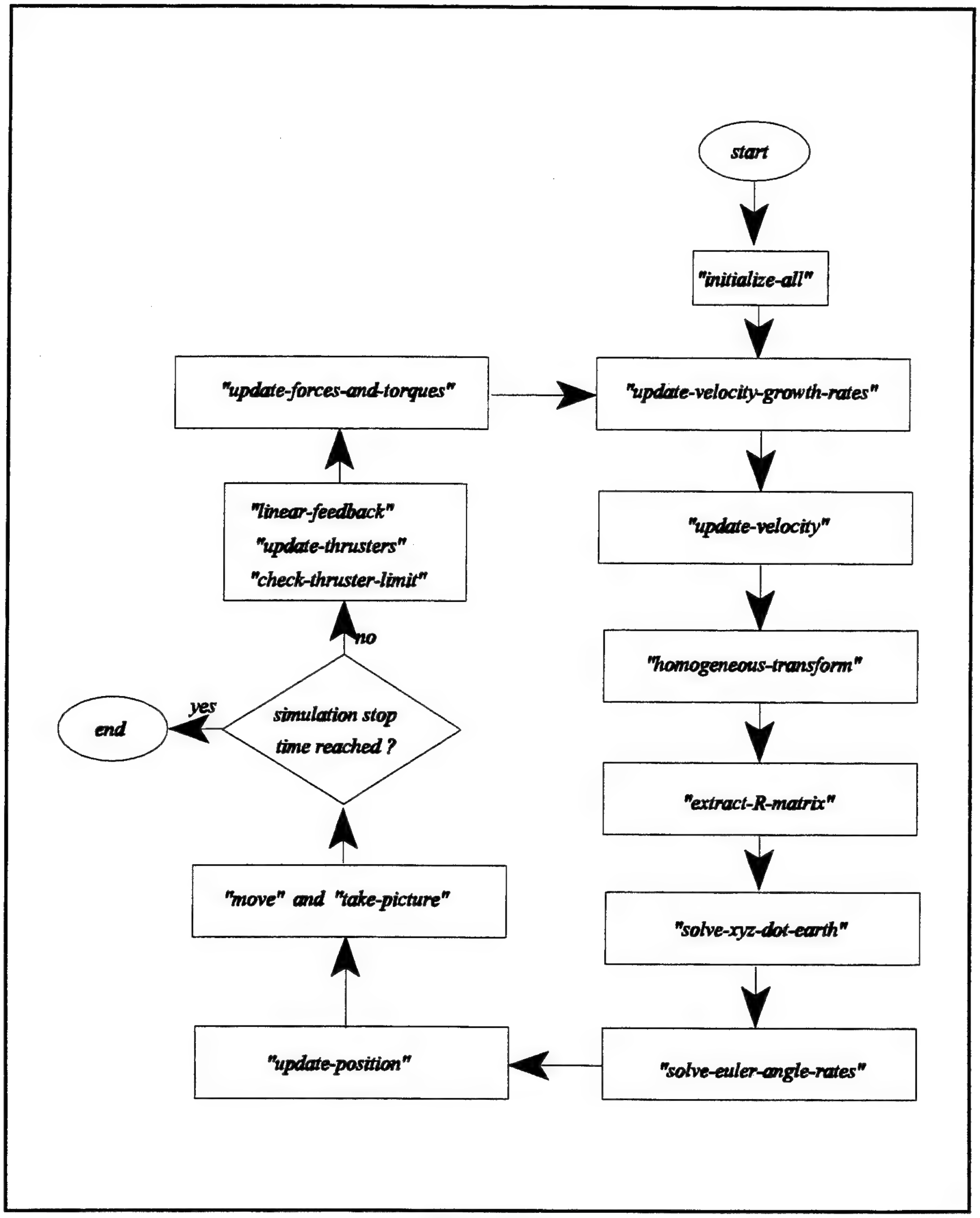

Figure 8

LISP Code Simulation Cycle Flowchart

definitions. Several details have been intentionally left out for clarity of the simulation process. The following subsections will briefly describe the functions named in the flowchart.

1. "initialize-all"

Before the simulation can run it must be initialized. This is accomplished by the function "initialize-all". The object class is created and the vehicle parameters (mass, added mass diagonal terms, buoyancy force, natural frequency and damping ratio) are set. Although the values used are those relative to the NPS Phoenix AUV, they can be set to anything. Default initial conditions and desired conditions are set as part of the initialization but can be changed prior to actually running the simulation. Finally, the windows for observing the AUV's movements are initialized.

2. "update-velocity-growth-rate"

This function uses the Newton-Euler dynamic equations discussed earlier to determine the body translational and rotational velocity growth rates. The results are the values from the mathematical model used to simulate the sensed data.

3. "update-velocity"

The Euler numerical integration method is used to obtain the body translational and rotational velocities from the Newton-Euler dynamic equations. This is accomplished by multiplying the velocity growth rates with the initialized time step and adding the result to the previous values of the velocities for the new body velocities. In order to keep the error accumulation to a reasonable value, a maximum time step of 0.01 seconds is used.

4. "homogeneous-transform"

The Homogeneous transformation matrix (H-matrix) is updated at each time step using this function. Equation (47) is implemented using the Euler angles and earth fixed coordinates to calculate the new Homogeneous transformation matrix.

5. "extract-R-matrix"

The direction cosine matrix (R-matrix) is extrapolated from the H-matrix. The R-matrix is used to transform the various vectors back and forth between the body coordinate system and the earth fixed coordinate system.

6. "solve-xyz-dot-earth"

This function uses the R-matrix to transform the body translational velocities to earth translational rates. These rates are then integrated using the Euler integration method.

7. "solve-euler-angle-rates"

The Euler angle rates are calculated by pre-multiplying the body rotational rates with the Non-orthogonal transformation matrix, which accomplished within this function.

8. "update-position"

Once the earth fixed translational and rotational rates have been determined using the previous two functions, the earth fixed coordinates and the Euler angles can be calculated. This is done by using the Euler numerical integration method. As in the "update-velocity" function, the rates are multiplied by the time step and then added to the previous coordinates and Euler angles to give the new position. Also, as in the "update-velocity" function, the time step used is 0.01 seconds to prevent an excessive error accumulation.

9. "move"

The H-matrix is applied to each node of the polygon figure using the calculated Euler angles and earth fixed coordinates. Once all the nodes have been translated and rotated to their new position the polygon figure is redrawn by applying the "take-picture" function.

10. "linear-feedback"

This function implements Equations (20) - (24), the control equations, to determine the forces and torques in the earth fixed frame. Since roll stability is incorporated into the physical design of the AUV, the torque about the x axis is neglected and assumed to be zero.

11. "update-thrusters"

Thruster forces are relative to the body coordinate system; hence, to use the earth fixed coordinate forces they must be transformed into body forces. This is accomplished by taking the inverse of the R-matrix, which happens to be its transpose as mentioned earlier, and pre-multiplying it to the earth forces. The Euler angles are used in the torque equations for the moments about the y and z axes. These body forces and torques are then pre-multiplied by the thruster coefficient matrix to obtain the required thruster forces. The required thruster forces are compared against the limits of the thrusters with the function "check-thruster-limit". If a required thruster force is greater than its limit, the autopilot simply saturates the thruster by commanding the maximum force that the thruster is capable of.

12. "update-forces-and-torques"

The Newton-Euler dynamic equations utilize the forces and torques resulting from the thruster commands given by the autopilot. Buoyancy effects are included in determining the forces and moments. This completes the simulation cycle.

E. SUMMARY

The philosophy behind the graphical simulation of the AUV has been presented. An explanation of the mathematical model used to simulate the AUV's dynamics which are applied to the designed autopilot was discussed. Several transformations and their purposes have also been presented. These transformations include the Homogeneous Transformation, Euler Angle Transformation and Non-orthogonal transformation. Finally, the essential code segments for implementing the autopilot design and simulating the responses to the autopilot both numerically and graphically have been presented. The next chapter discusses the results obtained from the autopilot design.

VI. SIMULATION RESULTS

A. INTRODUCTION

Since this autopilot design has not been incorporated into the NPS Phoenix AUV, actual data from the vehicles response could not be collected. Only the results of the simulation are presented. The only comparisons discussed are those between the initial design, which does not include added mass effects, and the second design, which includes only the diagonal terms of the added mass effects. As a means to compare the numerical values produced by the LISP code implementation of the autopilot, a MATLAB program was developed in parallel; however, the MATLAB code does not include the quadratic terms of the Newton-Euler dynamic equations. First the results of the MATLAB code implementation of the autopilot are discussed followed by the LISP code results. The initial conditions and desired conditions for the simulations are listed in Table 3. All simulations were run for 300 seconds, Figures 9 - 38 at the end of this chapter illustrate the results.

Simulation	Initial Conditions						Desired Conditions					
Run #	x	y	z	Ψ	θ	ϕ	x_o	y_o	z_o	Ψ_o	θ_o	ϕ_o
1	0	0	0	0	0	0	15	10	5	3.14	0	0
2	0	0	0	0	0	0	15	0	5	3.14	0	0
3	15	10	5	3.14	0	0	-15	-10	0	0	0	0
4	0	0	0	0	0	0	15	10	5	3.14	0.14	0
5	0	0	0	0	0	0	15	10	5	3.14	0.26	0

Table 3

Simulation Conditions

B. MATLAB CODE SIMULATION RESULTS

The MATLAB source code developed to compare against the LISP source code results is located in Appendix B. Figures 9 - 18 show the results of the MATLAB simulation where the solid lines represent the product of the first design which did not include the added mass effects. The dotted lines indicate the values calculated including the diagonal terms of the added mass effects. As can be seen in these figures, the end results of both designs are the same with only slight deviations between the two designs. Studying these figures it can be seen that the diagonal terms of the added mass effects are minimal.

C. LISP CODE SIMULATION RESULTS

The LISP source code is included as Appendix A. Figures 19 - 32 illustrate the values calculated by the LISP source code. The first design, which did not include the added mass effects, produced excellent results and are depicted as the solid lines in Figures 19 -28. When the second design was developed and simulated it performed as expected illustrated by the dotted lines in Figures 19 - 24, until a desired pitch angle other than zero radians was commanded. When a pitch angle was commanded the system produced extraneous results as can be seen in Figures 29 - 32. Since the second design of the MATLAB code yielded excellent results for all test cases it is assumed that the problem with the implementation of the second design in the LISP code is due to a programming error. Unfortunately, the bug has not been found.

Two differences between the MATLAB and LISP implementations are that the quadratic terms are neglected in the MATLAB code and the thruster limits were not incorporated into the MATLAB code. The latter is only relevant when the pitch angle is greater than 8.5 degrees, which was determined by studying Figures 27 - 28 where the thrusters have saturated. Figures 33 - 38 show how closely the second design implemented with the LISP code follows the first design implemented with the MATLAB code. Recall that the first design in MATLAB does not include the quadratic terms of the Newton-Euler dynamic equations or the added mass effects, while the second design in LISP includes both. The exception being the second design in LISP where the problem with pitch exists as shown in Figures 29 - 32.

D. CONCLUSIONS

The results of various simulation runs for both the MATLAB and LISP code implementations of the hover mode autopilot have been presented. The simulations of the first design for both MATLAB and LISP showed favorable results. Simulations for the second design were also favorable from the MATLAB code. The LISP code simulations for the second design were as expected except for when a pitch angle was commanded. Although this remains unresolved, a significant advancement in the current autopilot implementation has been achieved. It is concluded that due to the relative slow speed of the AUV during hovering and short transits the quadratic terms are negligible, that is, the linearized analysis predicts results well. This is not always true. For example, in walking machine postural control much more rapid response is needed to prevent falling over; that is, the quadratic terms are significant as shown in Figure 5b of (McGhee, 1986). In this example, linear design produces stability but does not predict quadratic coupling between attitude and altitude.

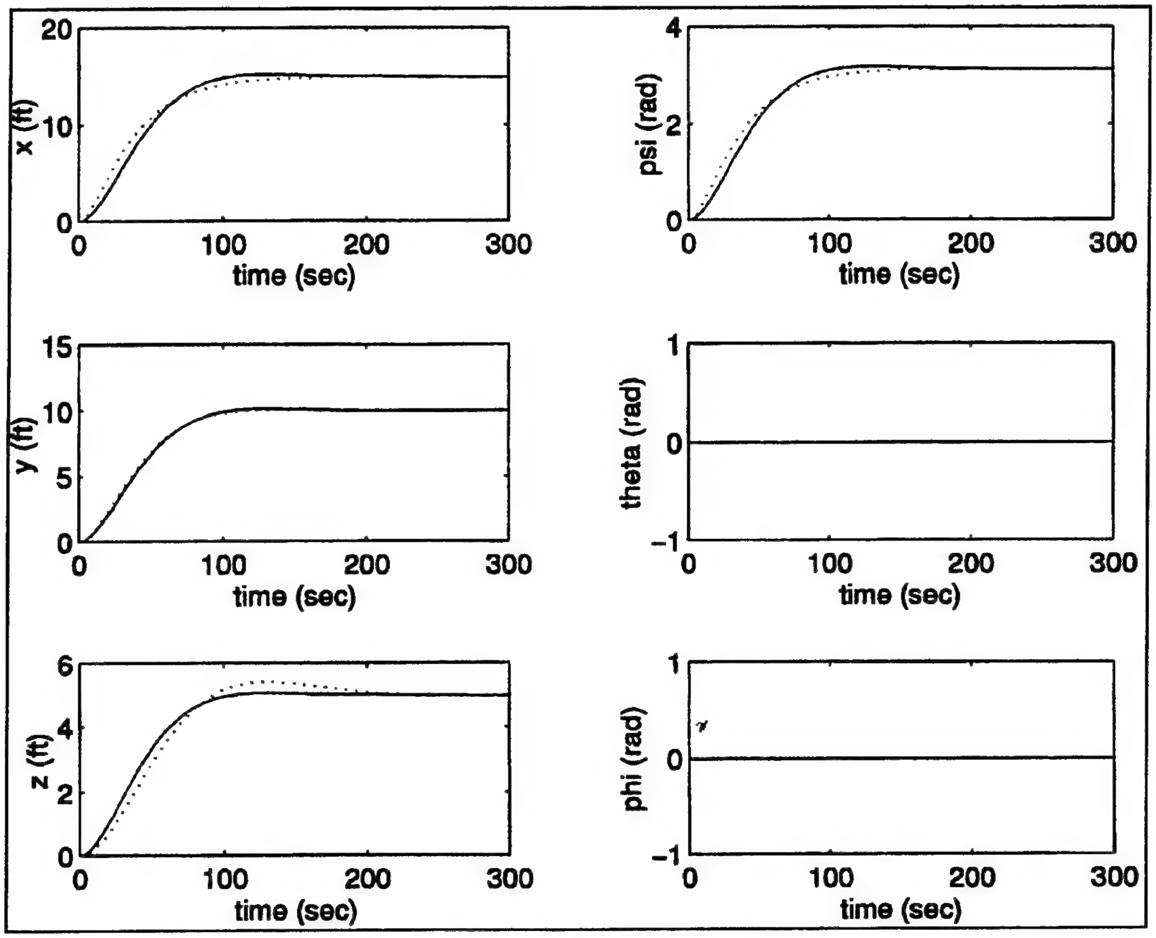

Figure 9

MATLAB Posture Results Simulation Run #1
Dotted Line Includes Added Mass Effects Solid Line Does Not

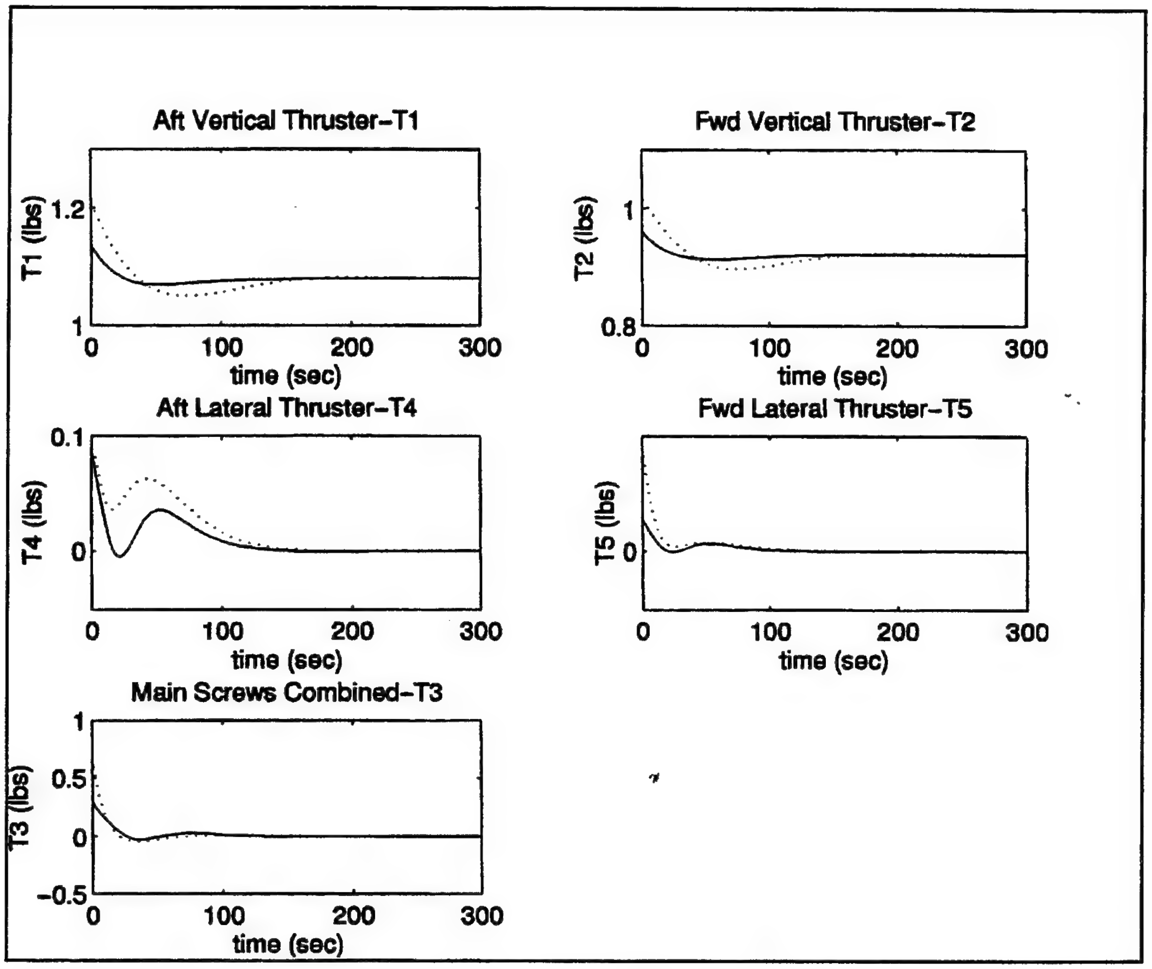

Figure 10

MATLAB Thruster Force Results Simulation Run #1
Dotted Line Includes Added Mass Effects Solid Line Does Not

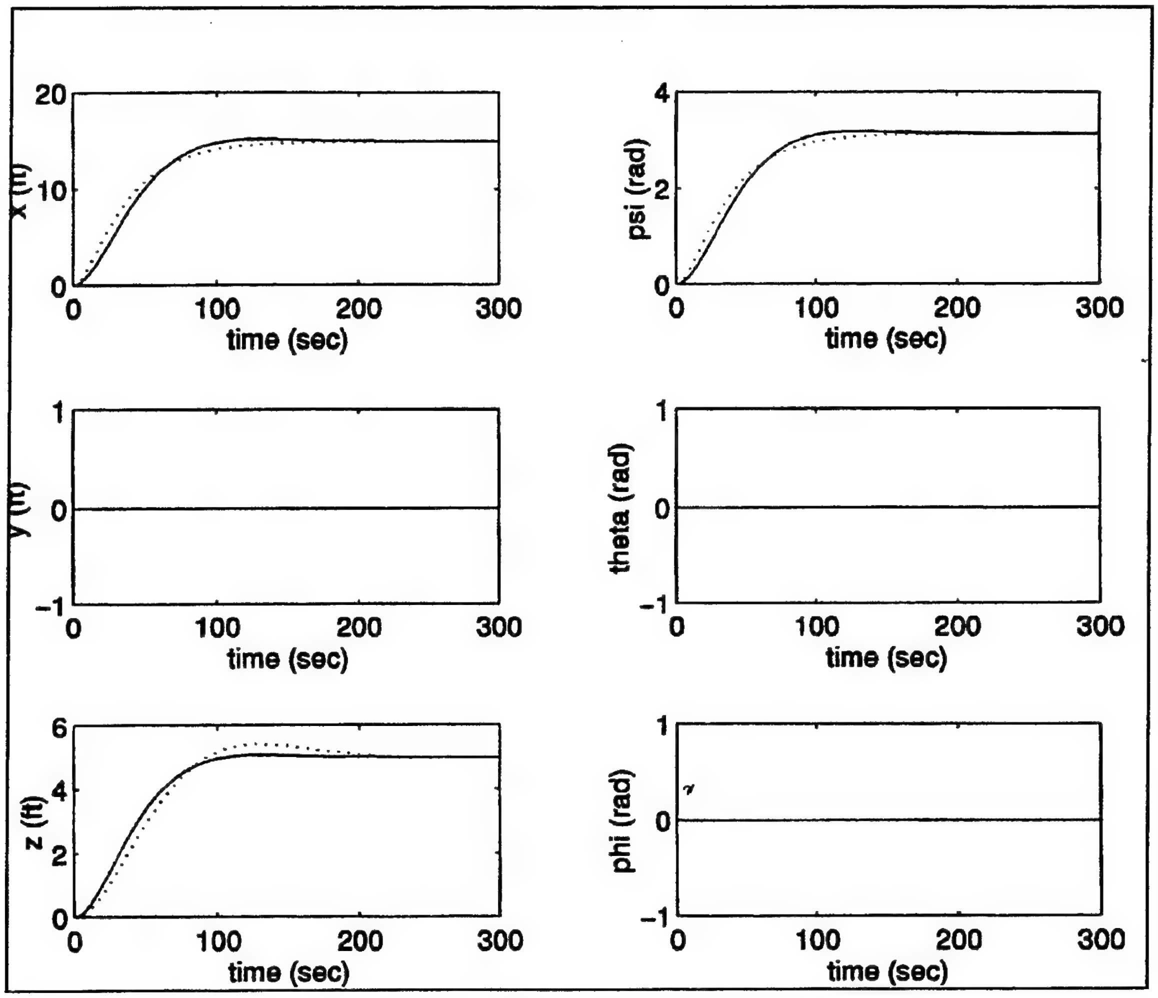

Figure 11

MATLAB Posture Results Simulation Run #2
Dotted Line Includes Added Mass Effects Solid Line Does Not

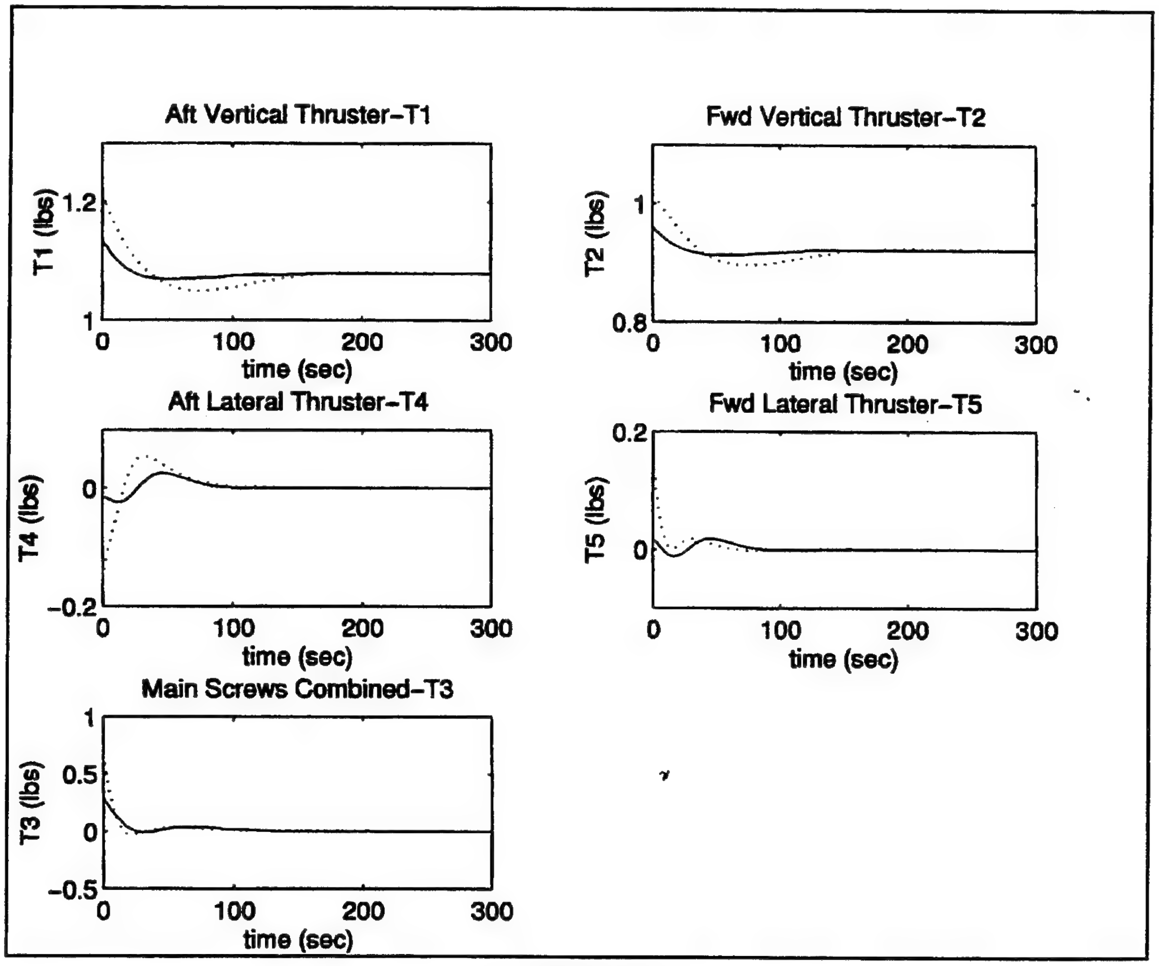

Figure 12

MATLAB Thruster Force Results Simulation Run #2
Dotted Line Includes Added Mass Effects Solid Line Does Not

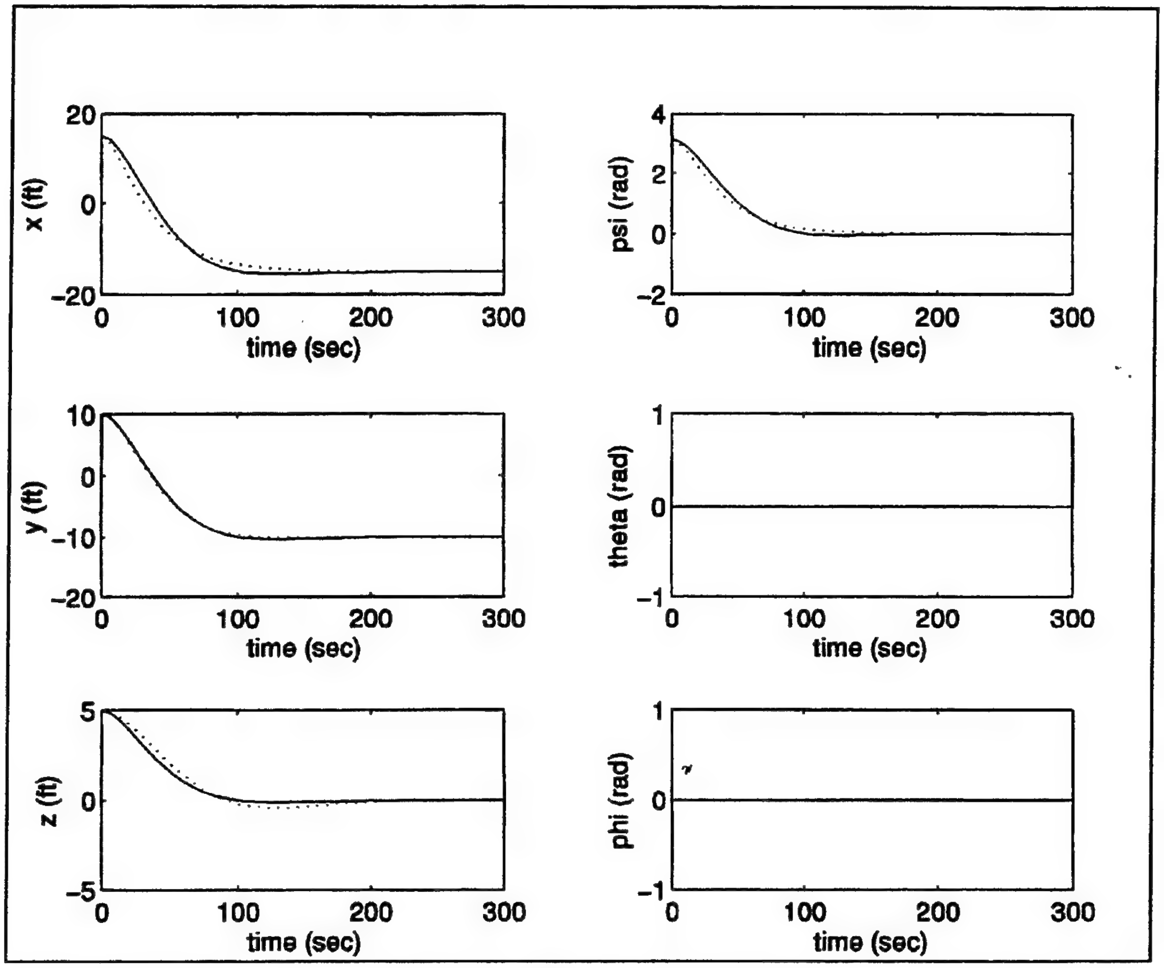

Figure 13

MATLAB Posture Results Simulation Run #3
Dotted Line Includes Added Mass Effects Solid Line Does Not

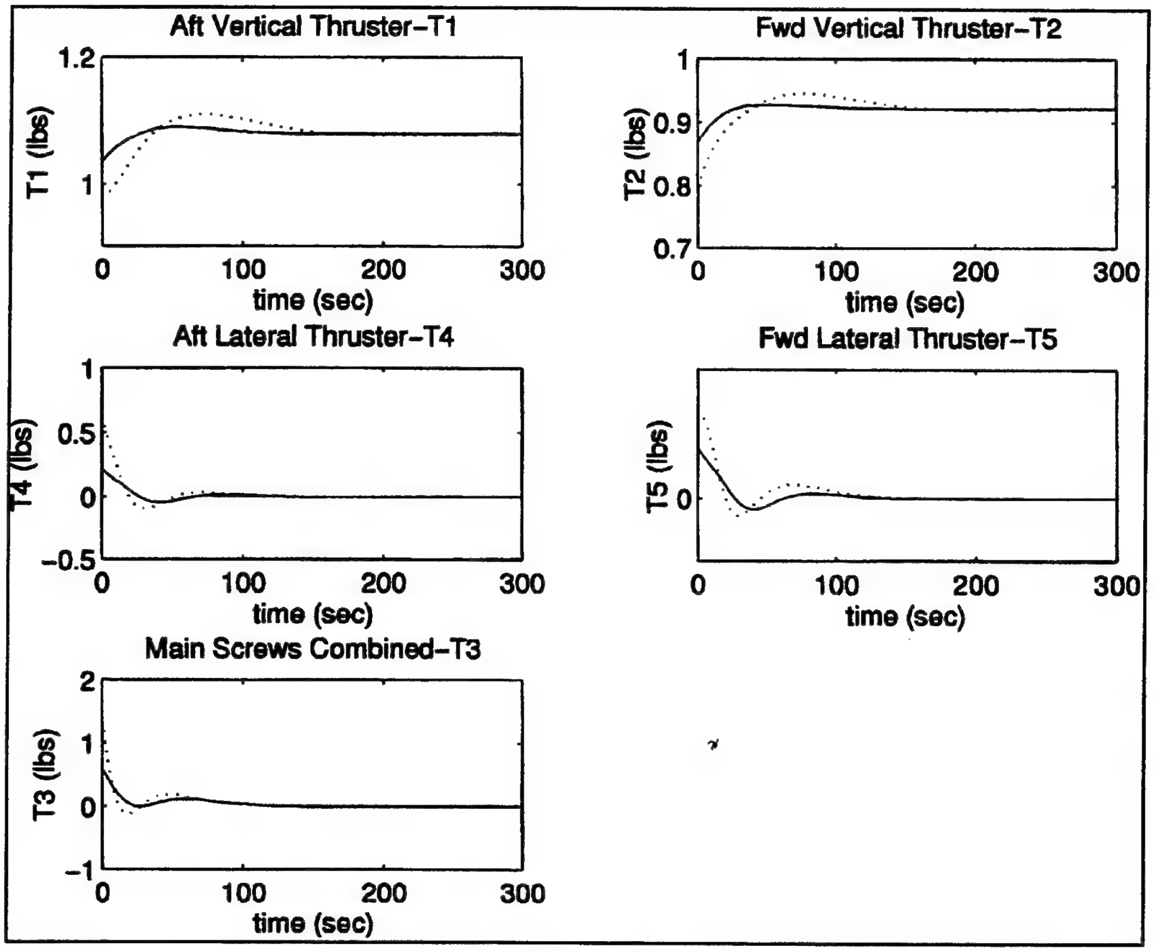

Figure 14

MATLAB Thruster Force Results Simulation Run #3
Dotted Line Includes Added Mass Effects Solid Line Does Not

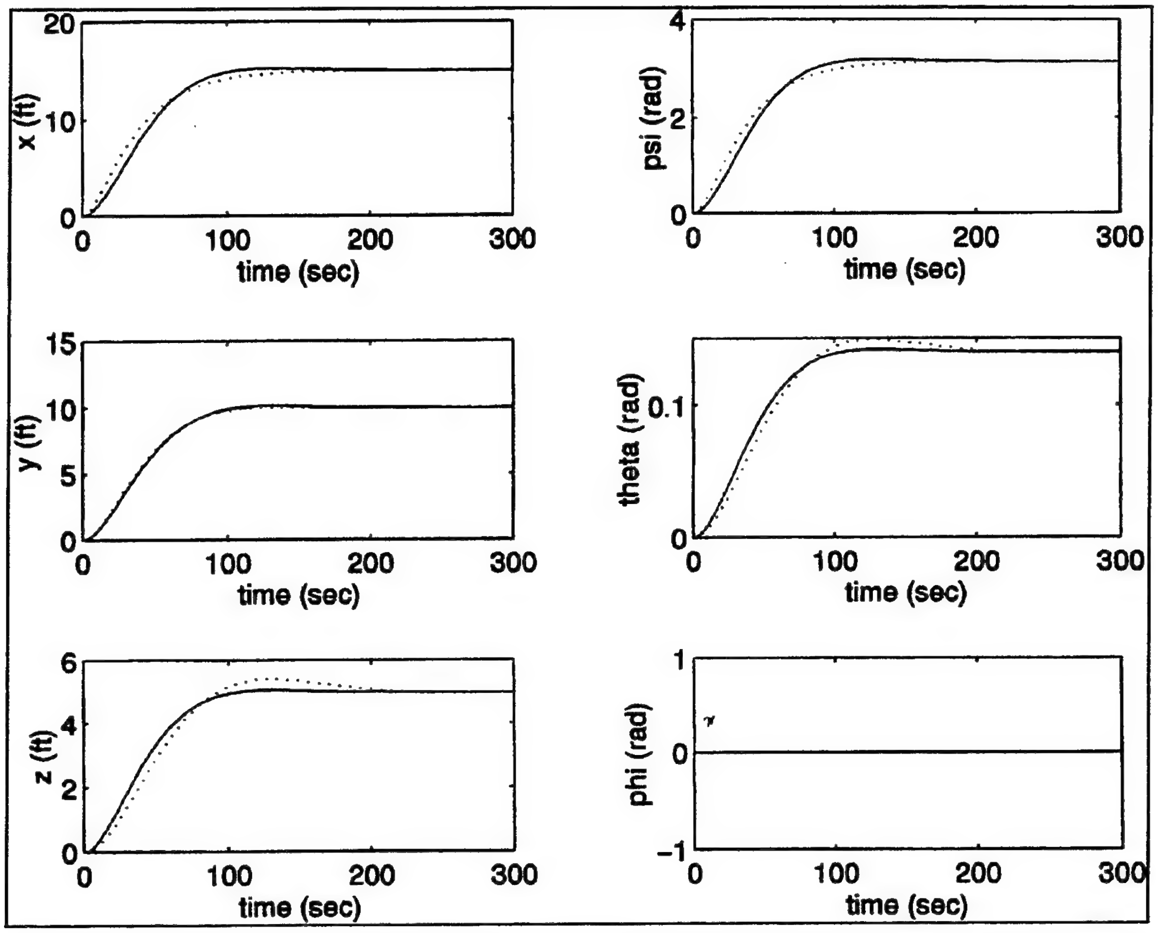

Figure 15

MATLAB Posture Results Simulation Run #4
Dotted Line Includes Added Mass Effects Solid Line Does Not

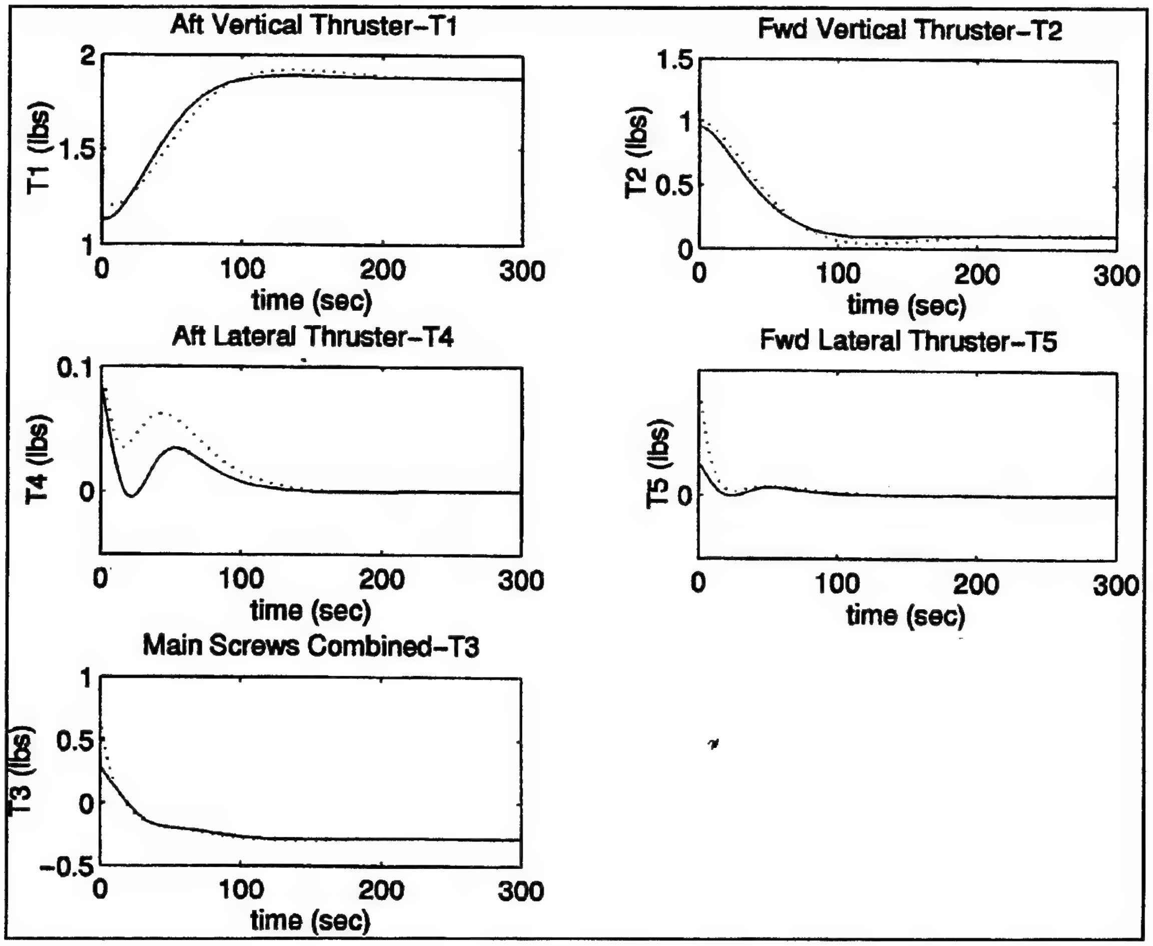

Figure 16

MATLAB Thruster Force Results Simulation Run #4
Dotted Line Includes Added Mass Effects Solid Line Does Not

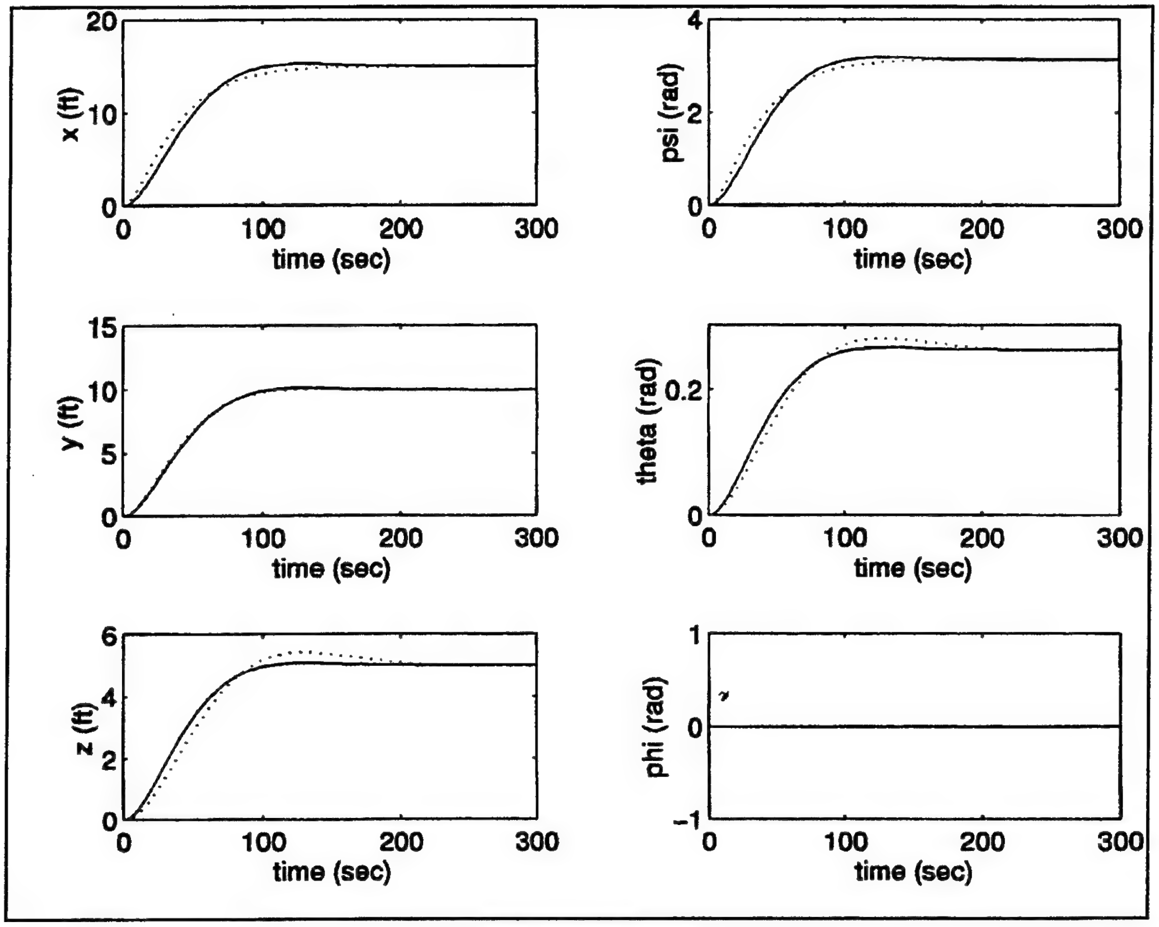

Figure 17

MATLAB Posture Results Simulation Run #5
Dotted Line Includes Added Mass Effects Solid Line Does Not

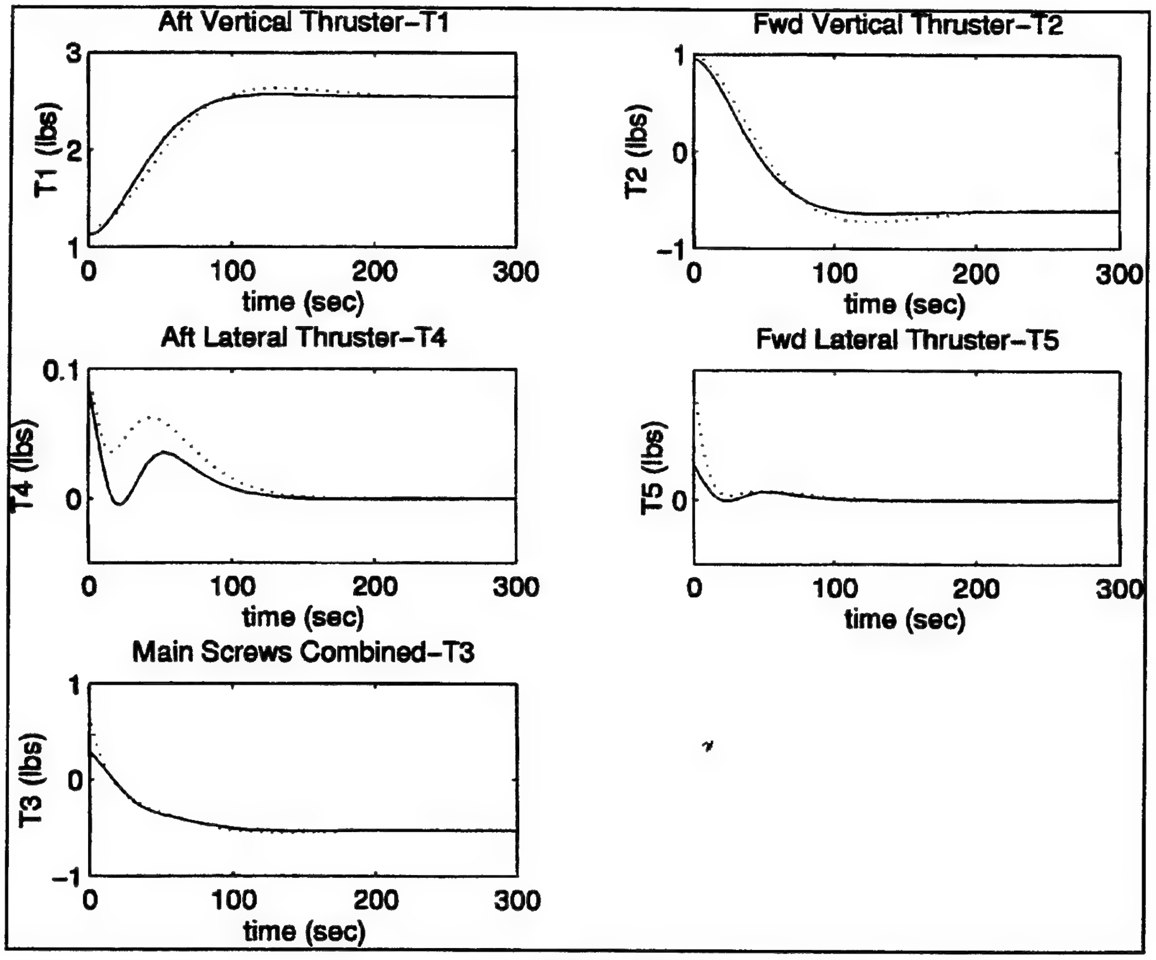

Figure 18

MATLAB Thruster Force Results Simulation Run #5
Dotted Line Includes Added Mass Effects Solid Line Does Not

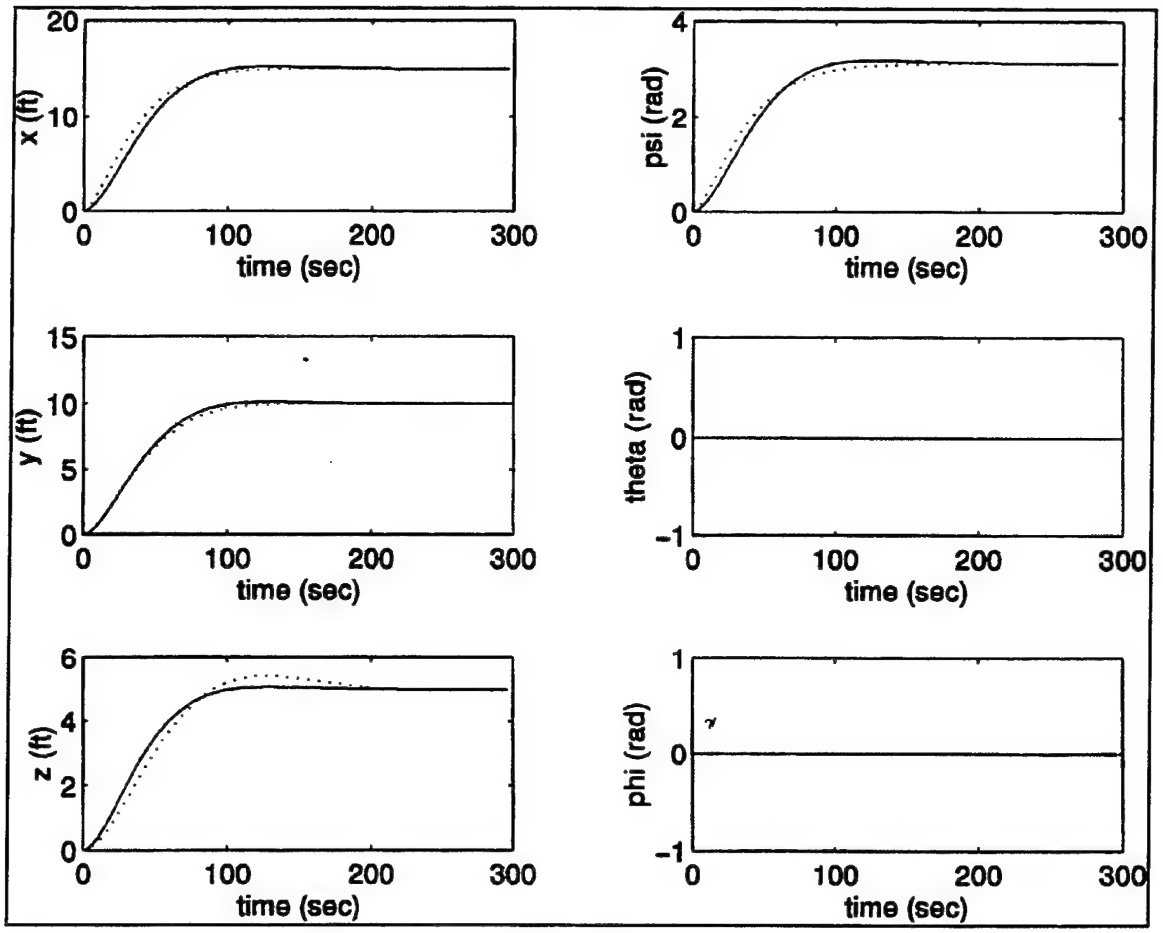

Figure 19

LISP Posture Results Simulation Run #1
Dotted Line Includes Added Mass Effects Solid Line Does Not

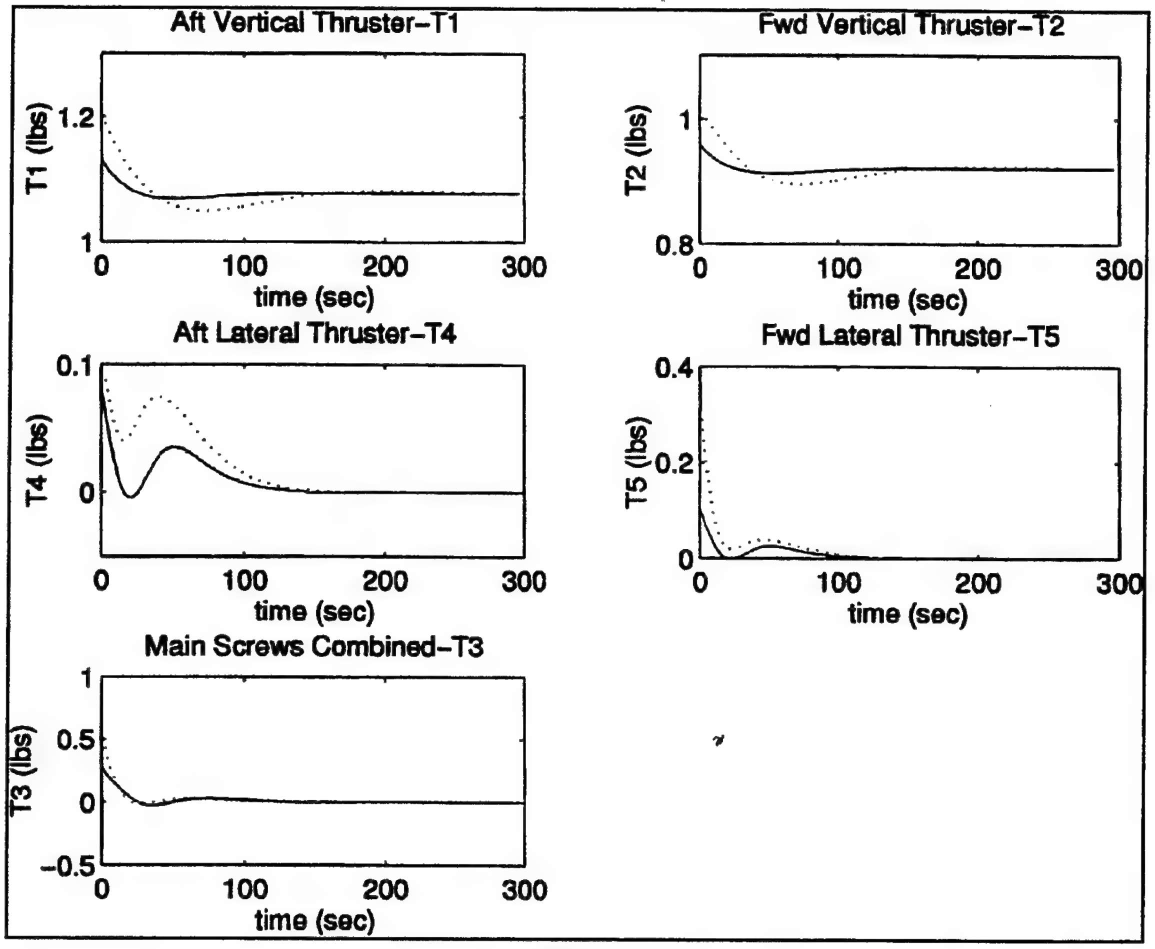

Figure 20

LISP Thruster Force Results Simulation Run #1
Dotted Line Includes Added Mass Effects Solid Line Does Not

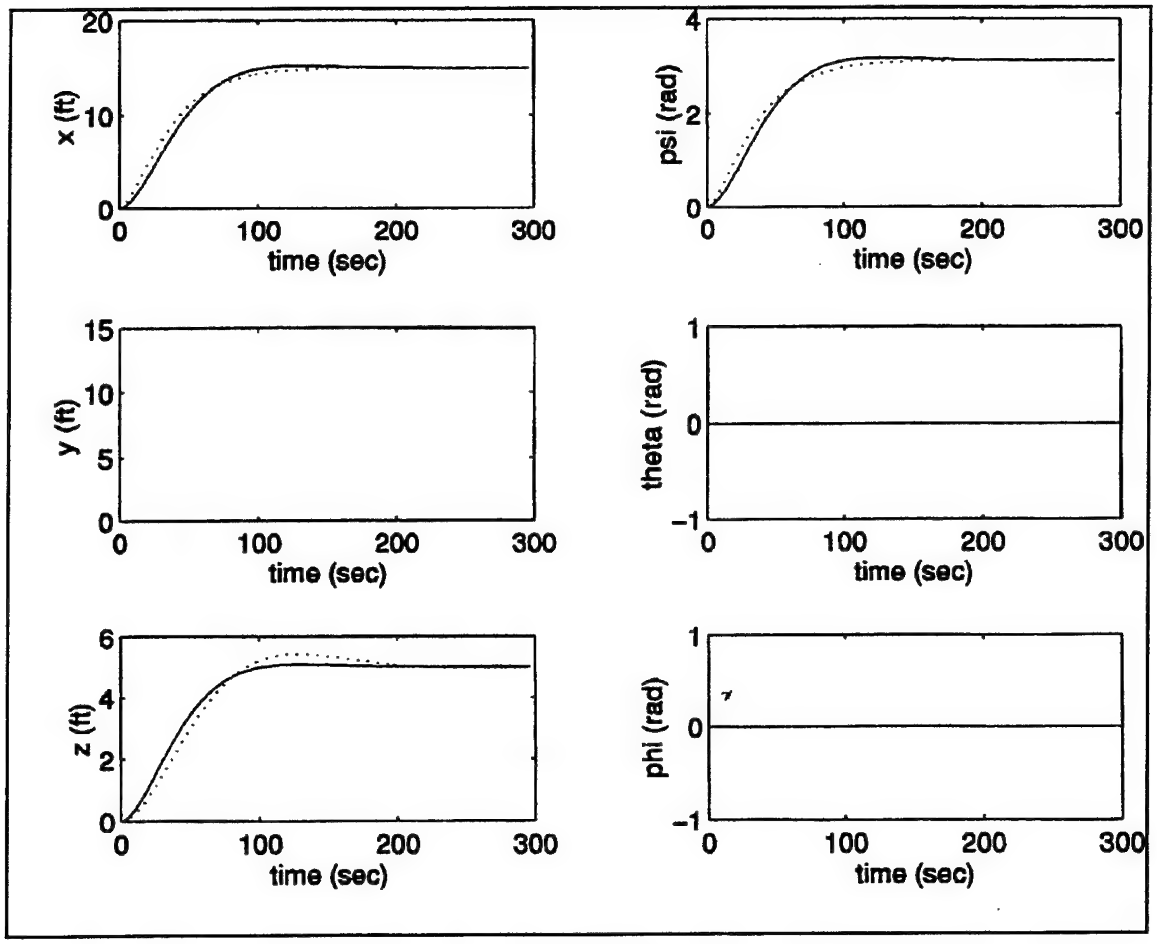

Figure 21

LISP Posture Results Simulation Run#2
Dotted Line Includes Added Mass Effects Solid Line Does Not

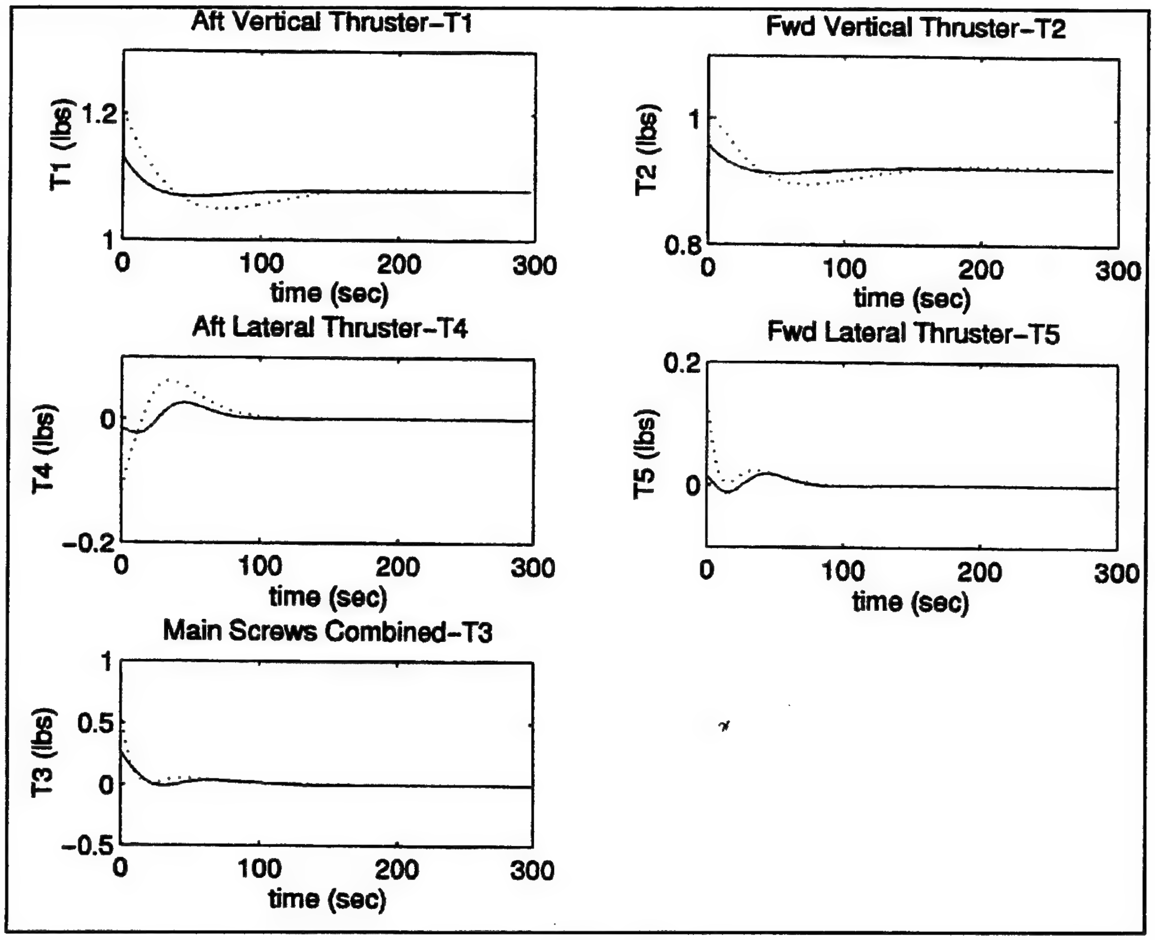

Figure 22

LISP Thruster Force Results Simulation Run #2
Dotted Line Includes Added Mass Effects Solid Line Does Not

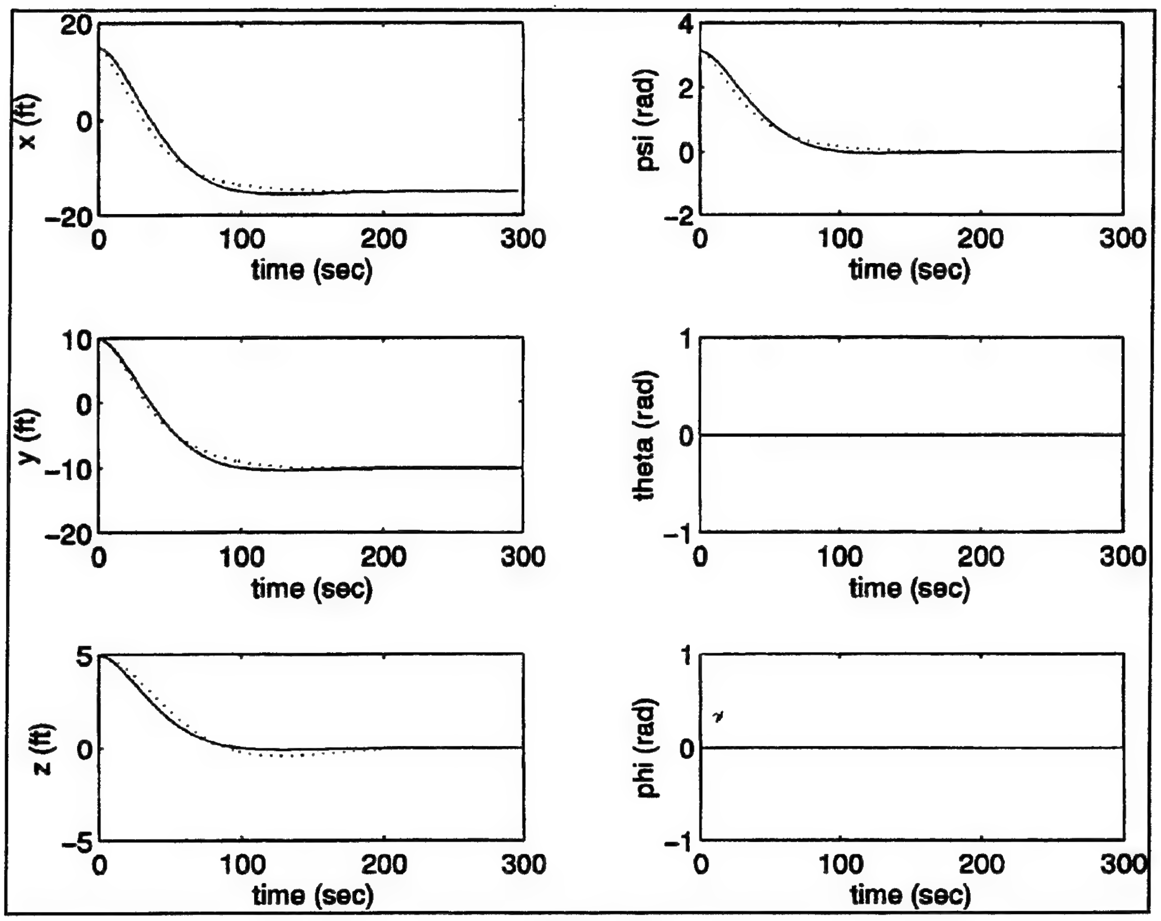

Figure 23

LISP Posture Results Simulation Run #3
Dotted Line Includes Added Mass Effects Solid Line Does Not

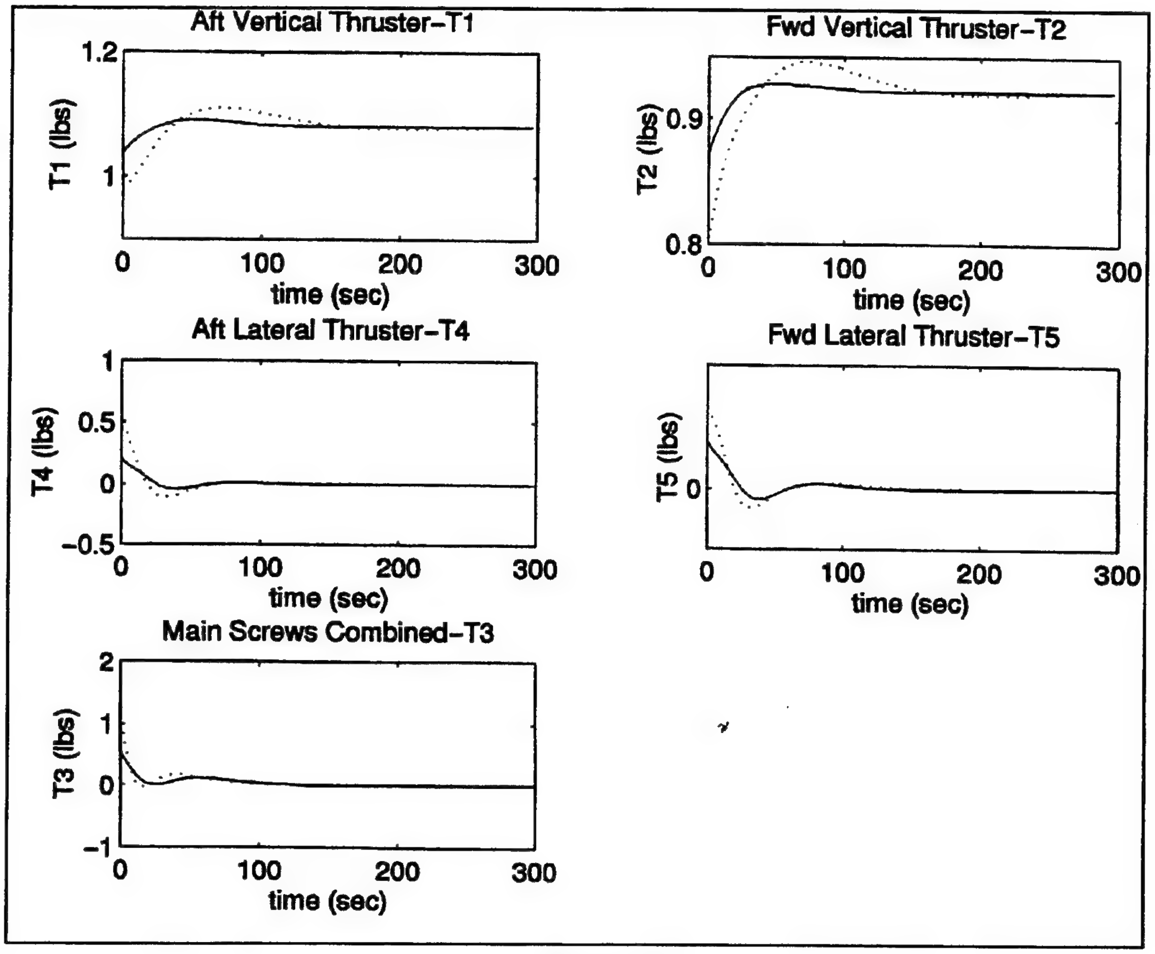

Figure 24

LISP Thruster Force Results Simulation Run #3
Dotted Line Includes Added Mass Effects Solid Line Does Not

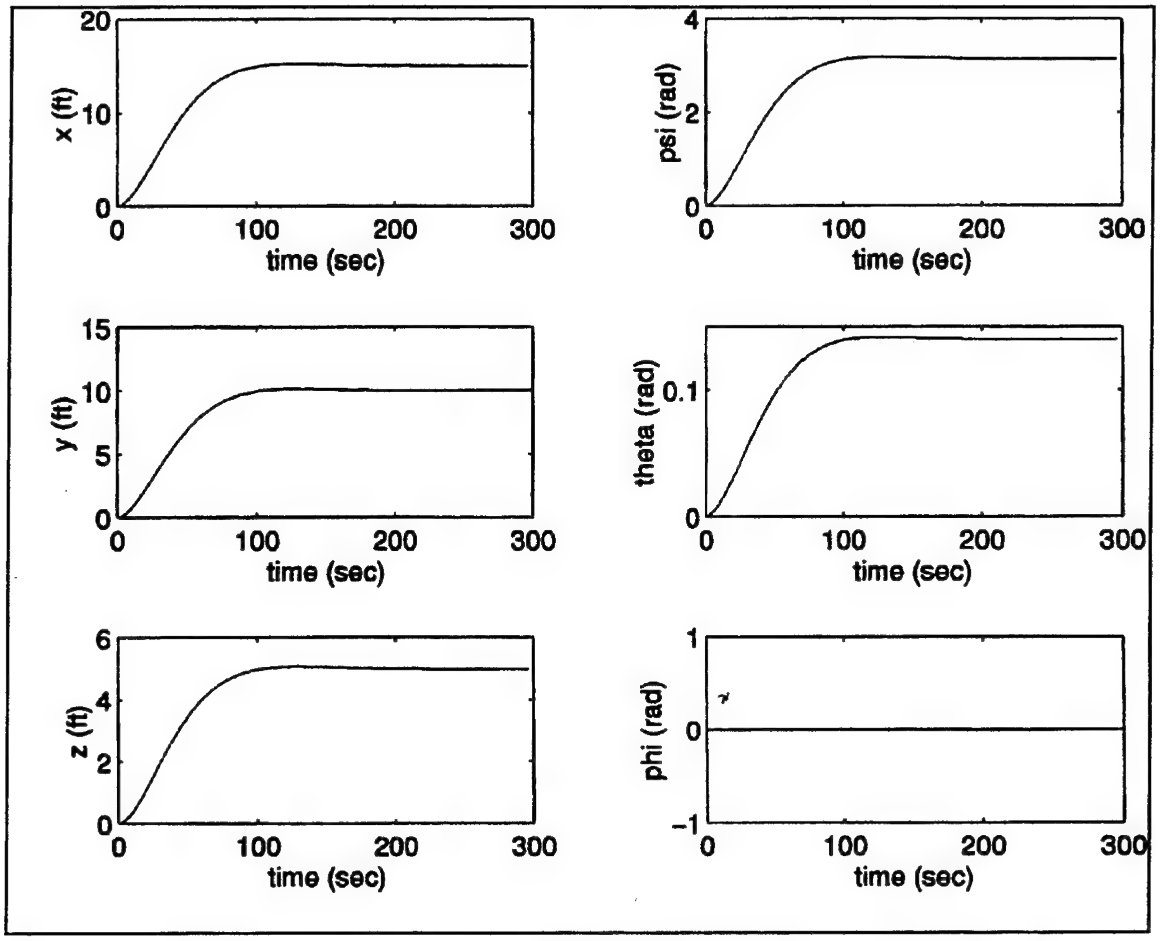

Figure 25

LISP Design-1 Posture Results Simulation Run #4

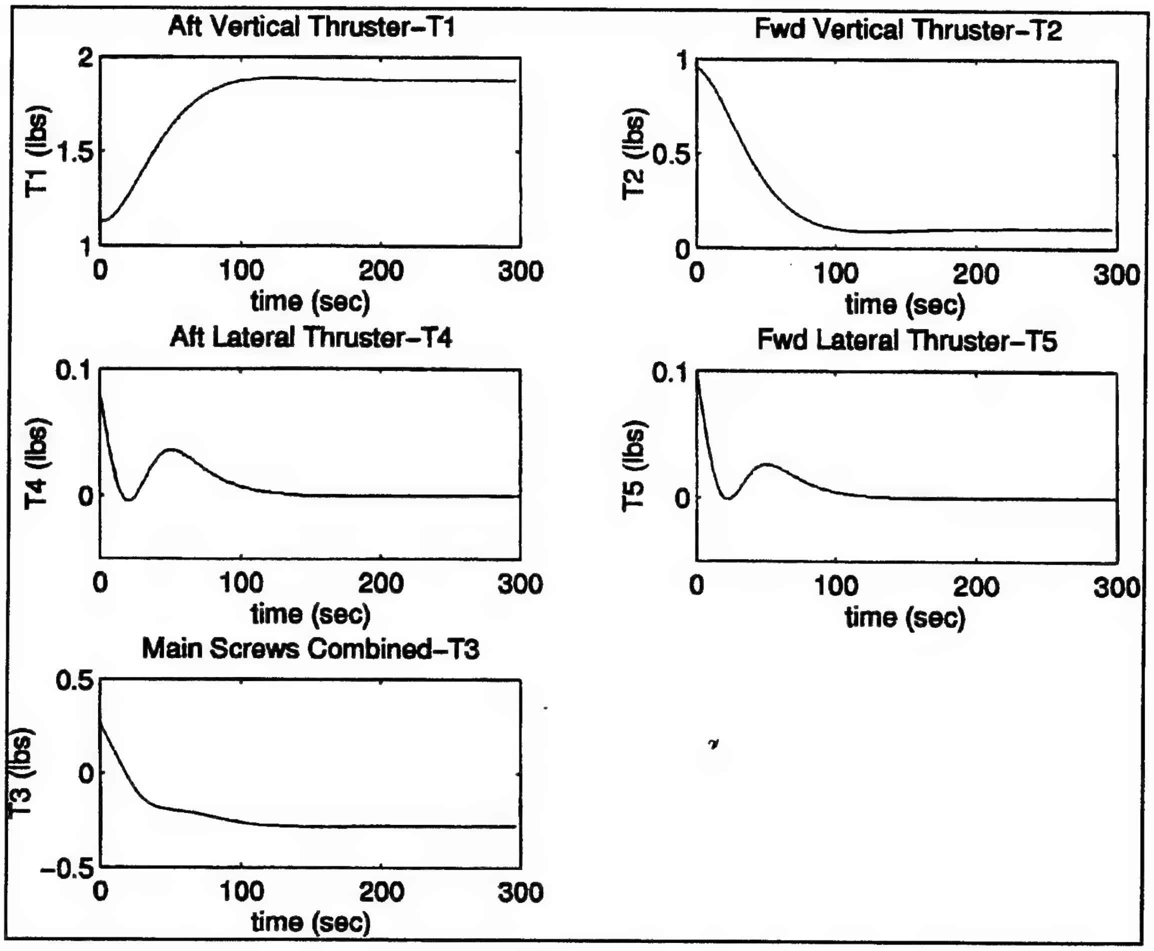

Figure 26

LISP Design-1 Thruster Force Results Simulation Run #4

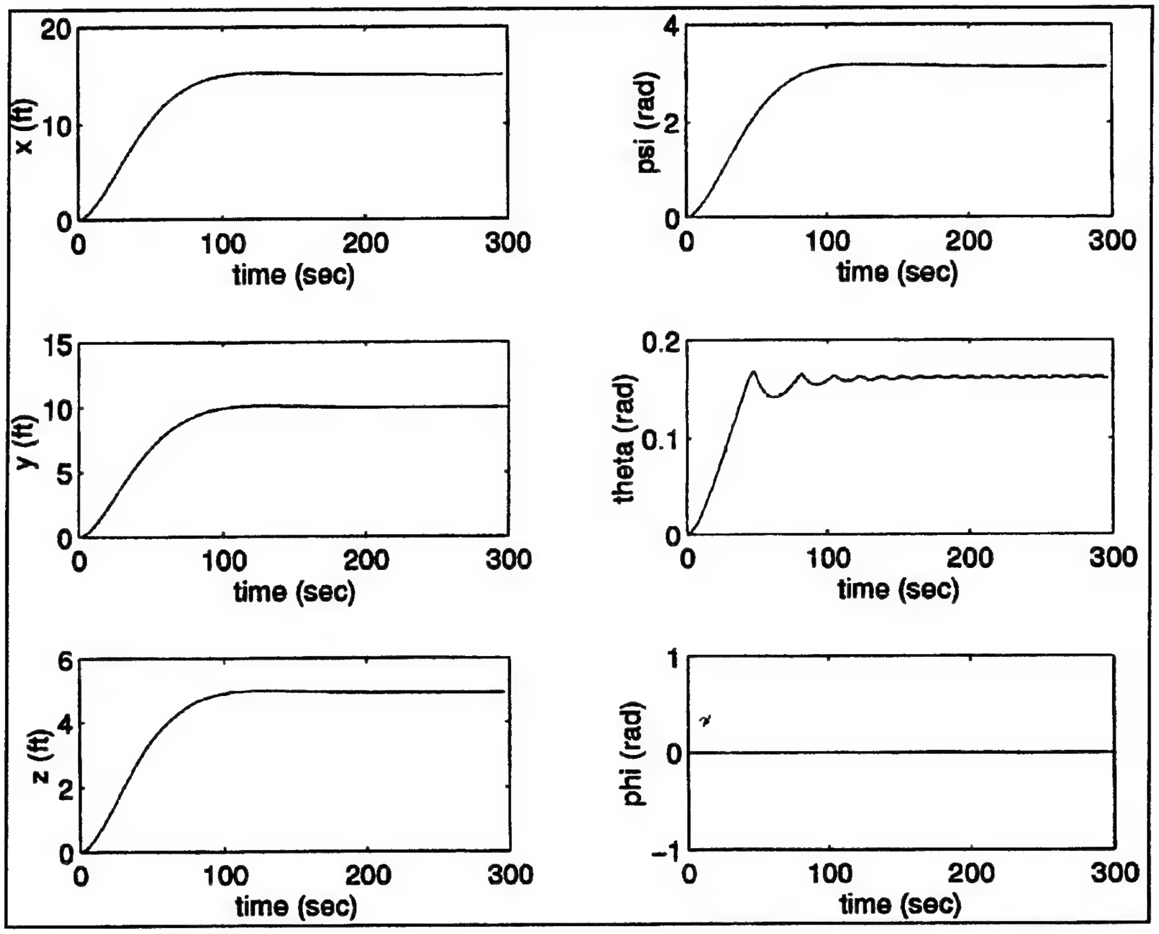

Figure 27

LISP Design-1 Posture Results Simulation Run #5

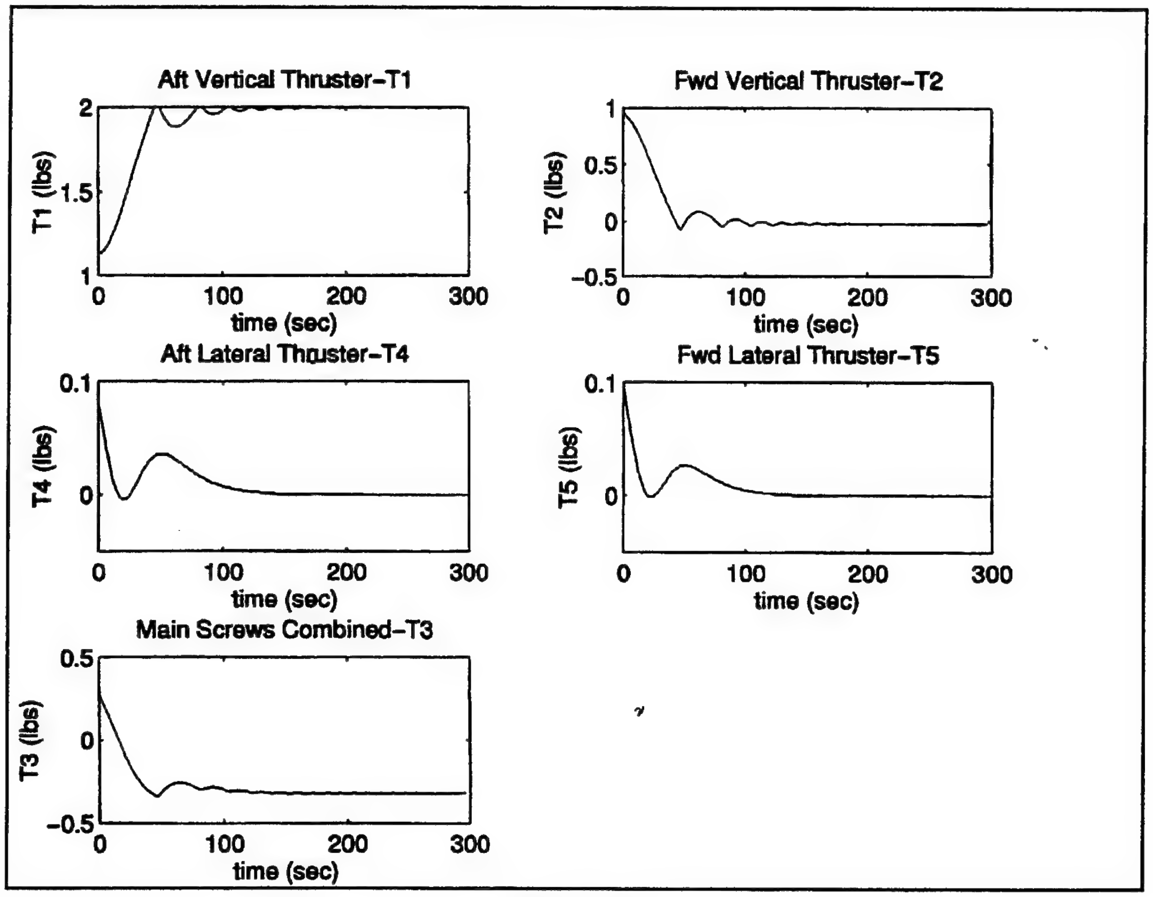

Figure 28

LISP Design-1 Thruster Force Results Simulation Run #5

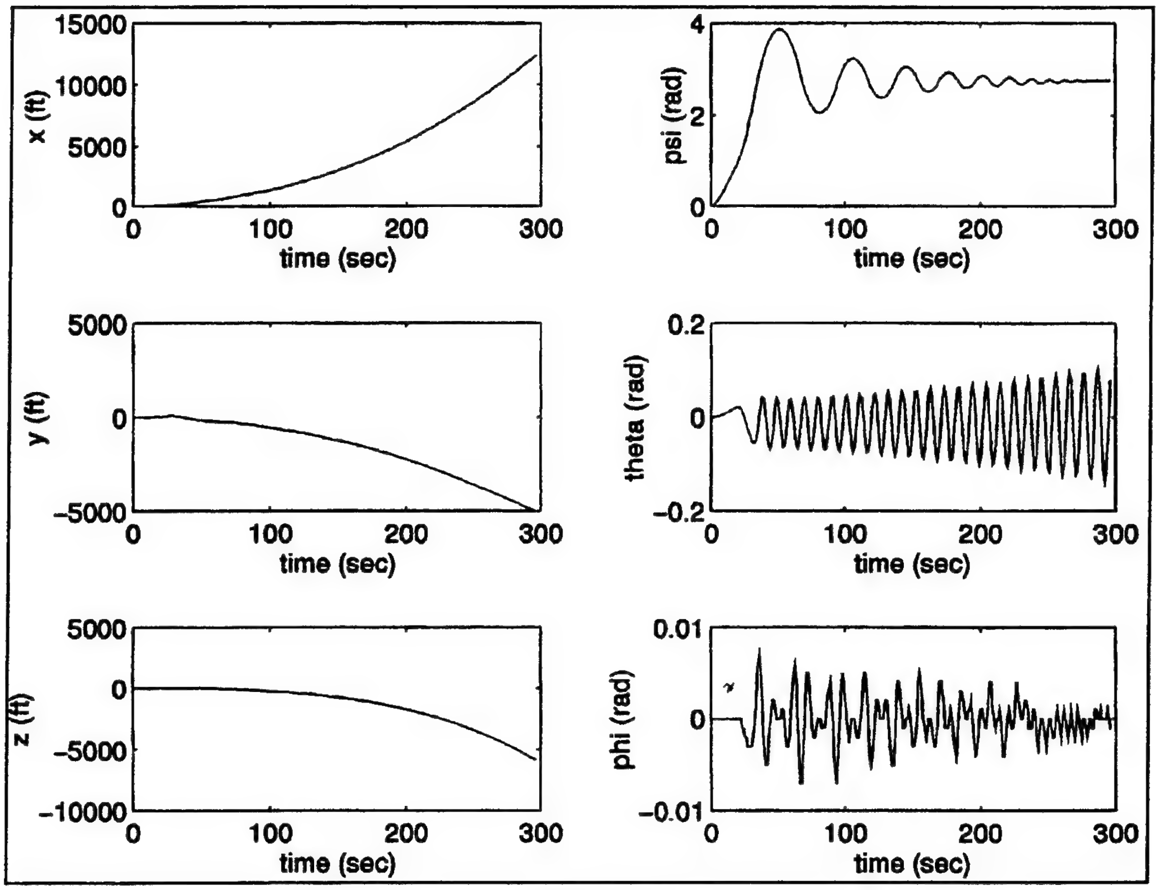

Figure 29

LISP Design-2 Posture Results Simulation Run #4

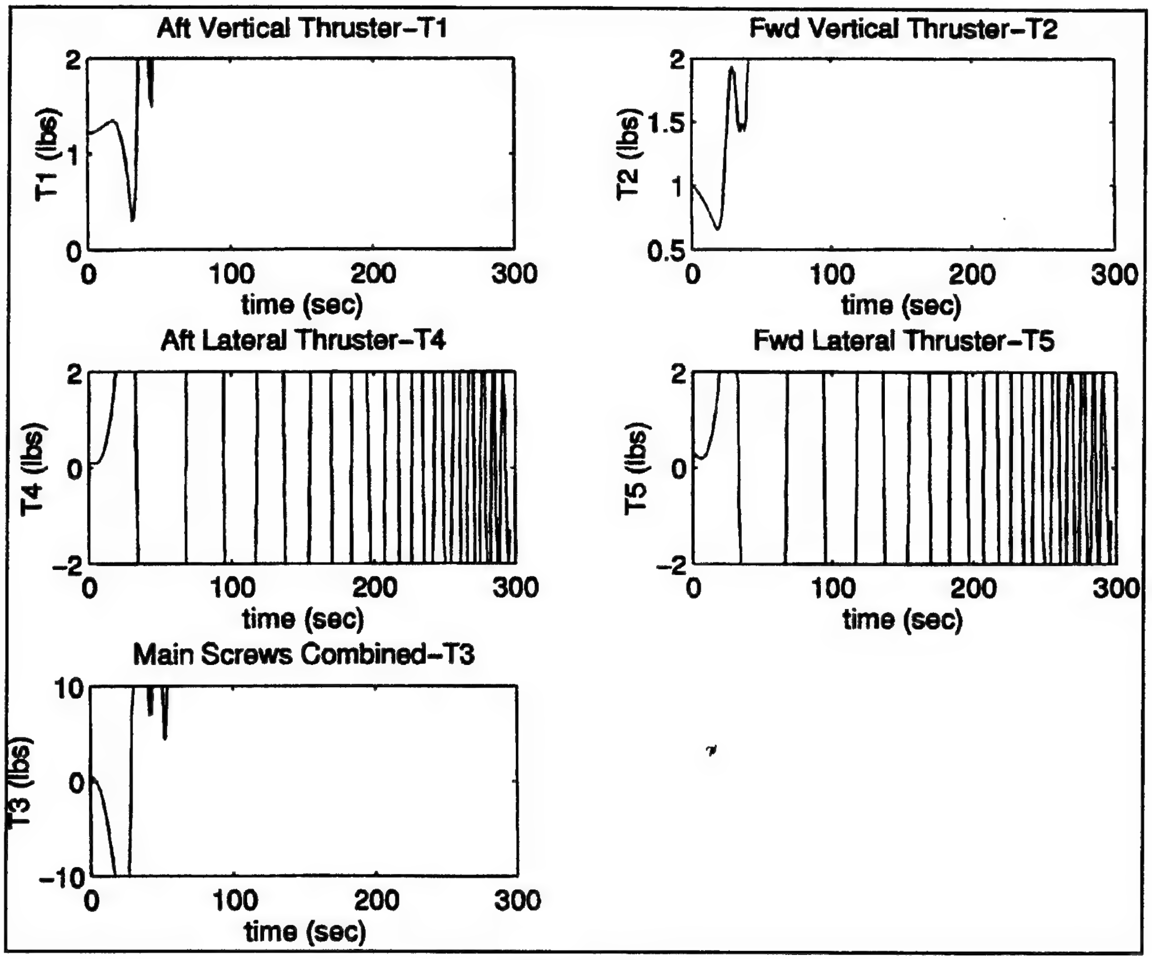

Figure 30

LISP Design-2 Thruster Force Results Simulation Run #4

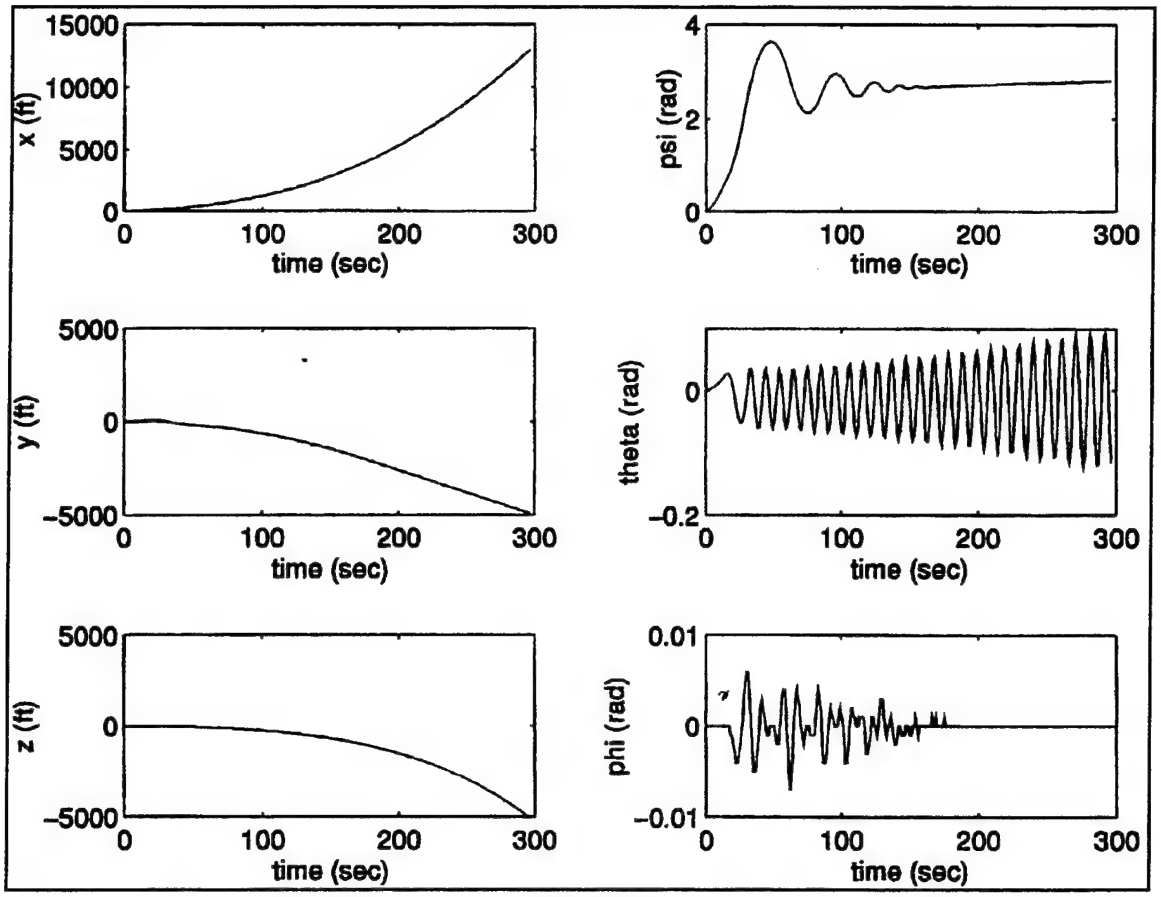

Figure 31

LISP Design-2 Posture Results Simulation Run #5

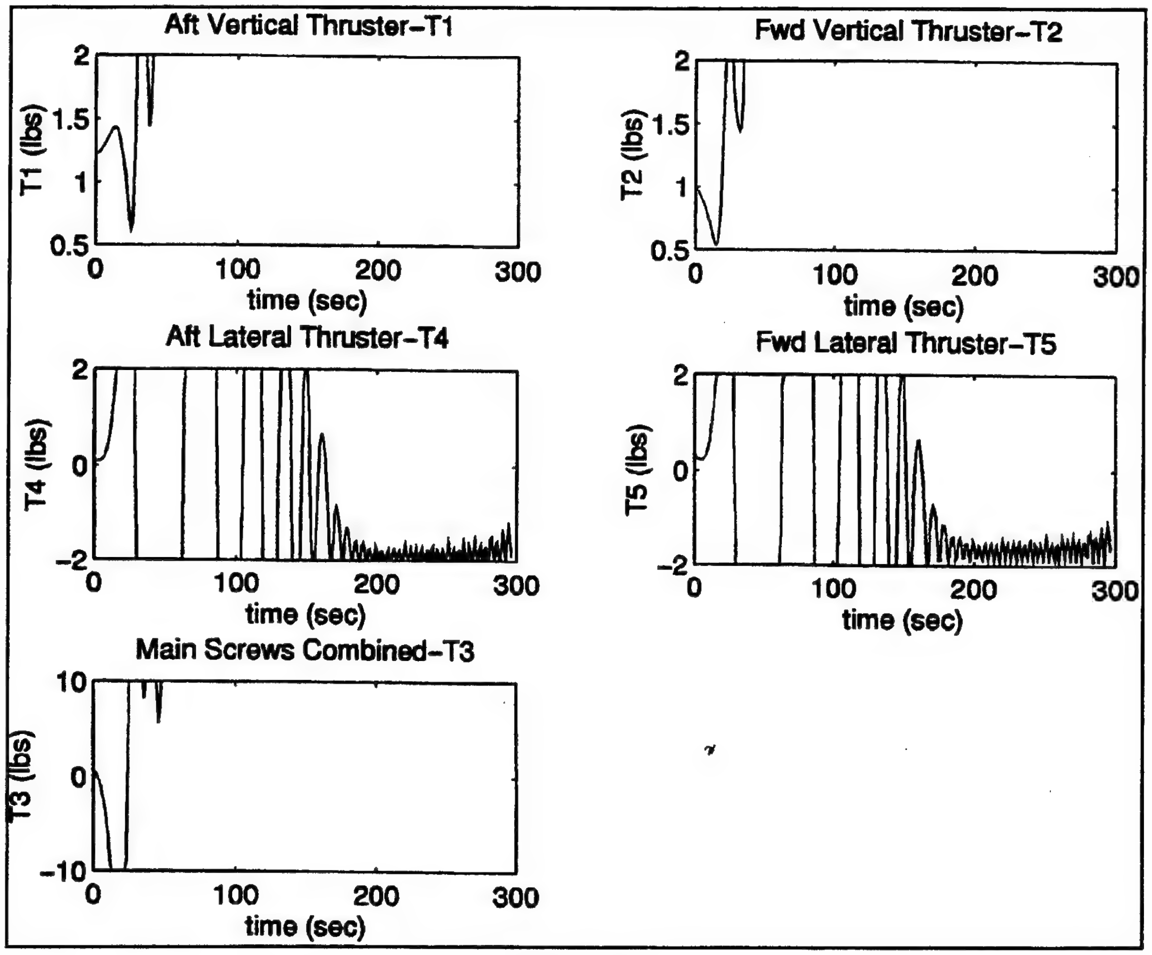

Figure 32

LISP Design-2 Thruster Force Results Simulation Run #5

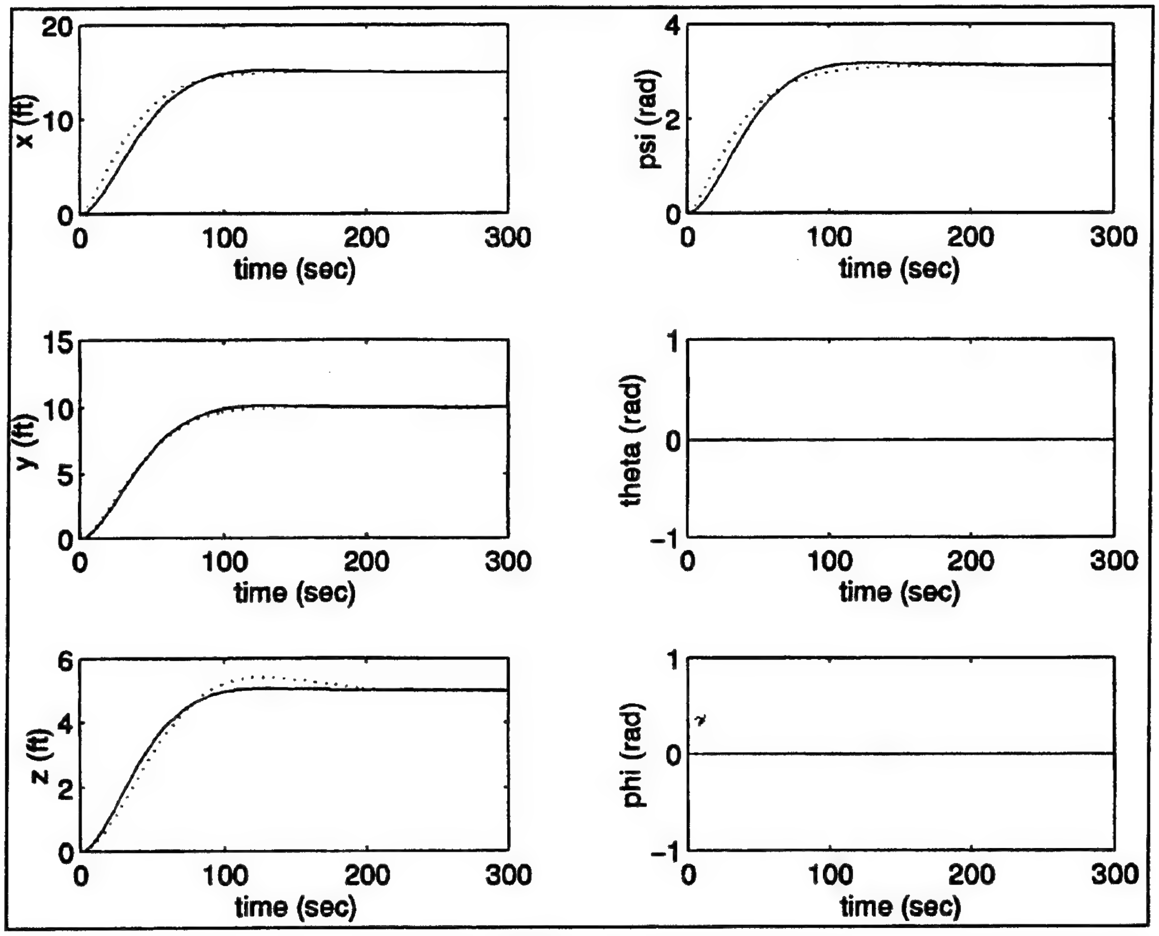

Figure 33

MATLAB Design-1 (Solid) vs. LISP Design-2 (Dotted) Posture Results Run#1

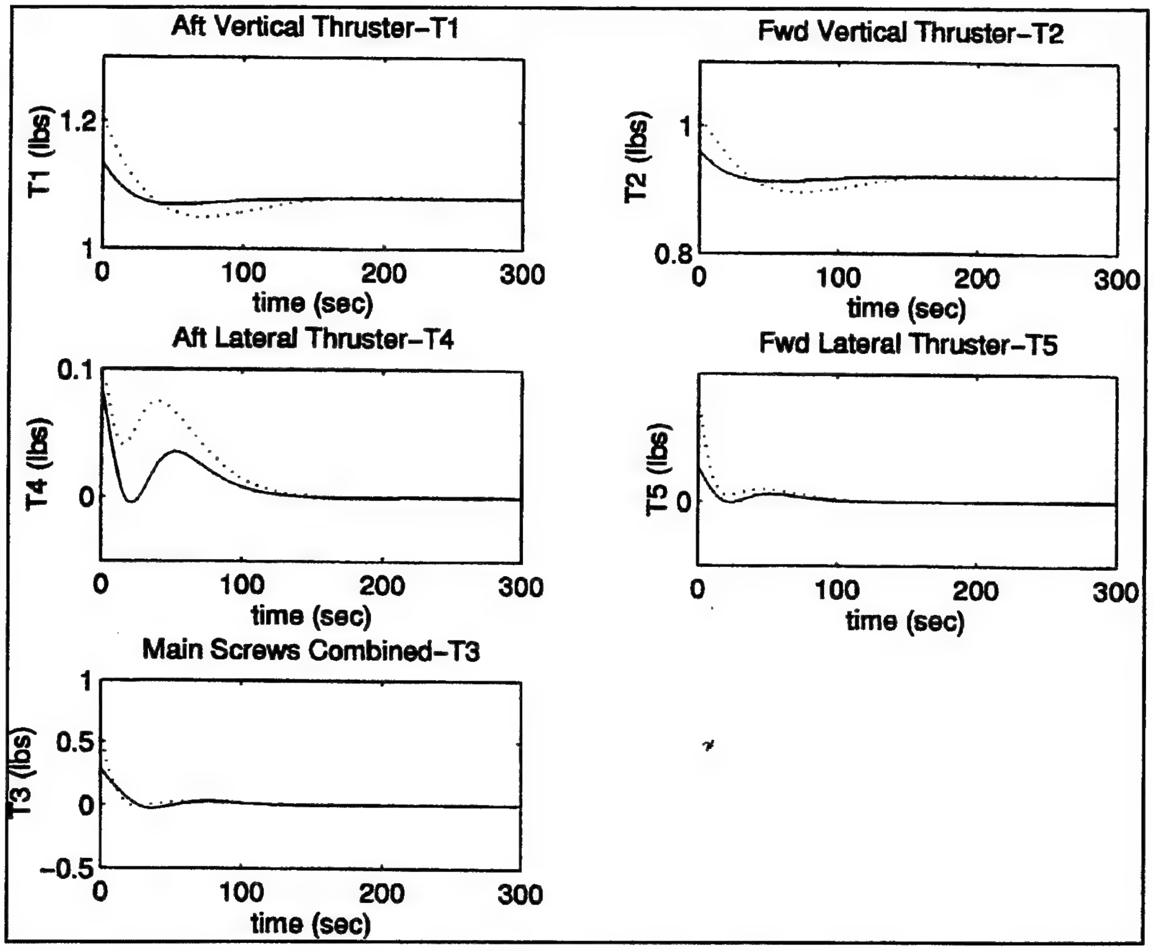

Figure 34

MATLAB Design-1 (Solid) vs. LISP Design-2 (Dotted) Thruster Force Results Run#1

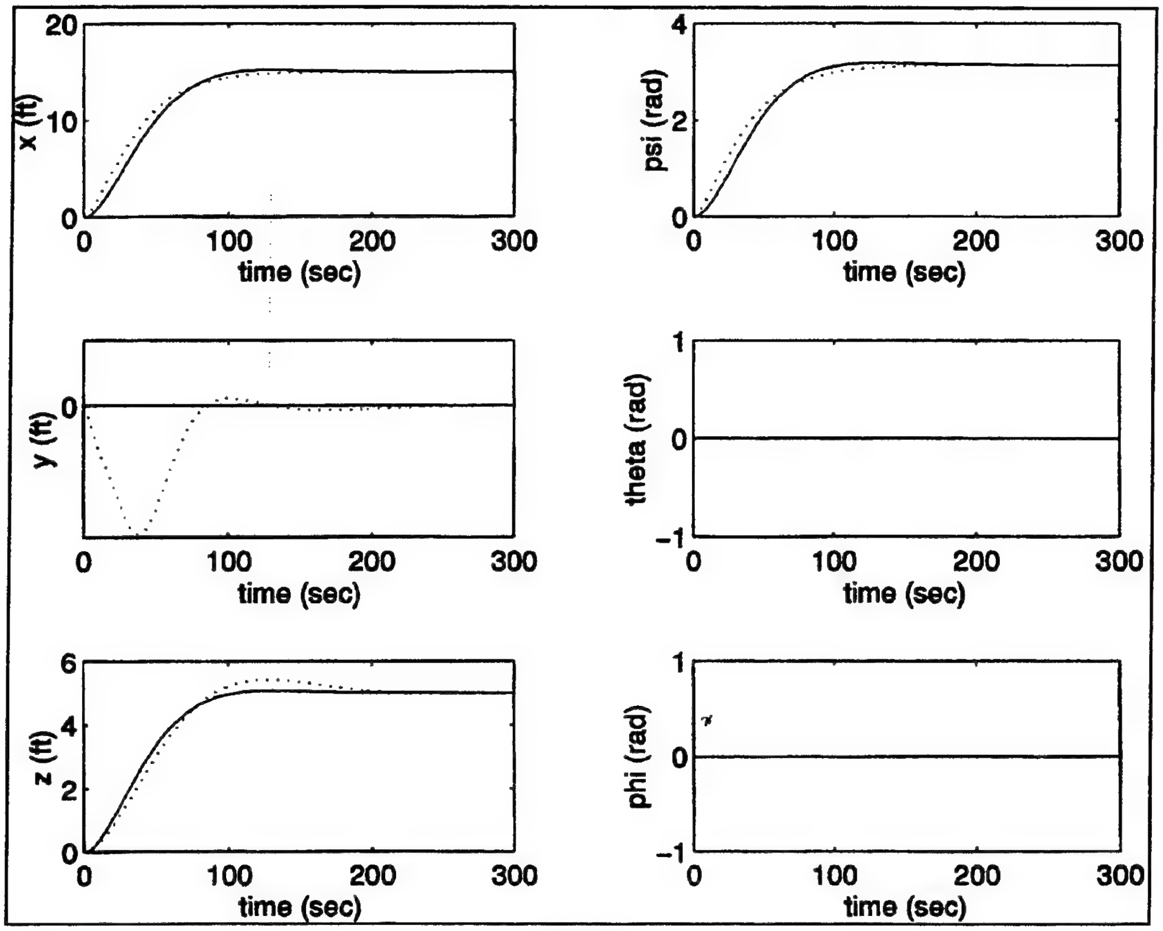

Figure 35

MATLAB Design-1(Solid) vs. LISP Design-2 (Dotted) Posture Results Run#2

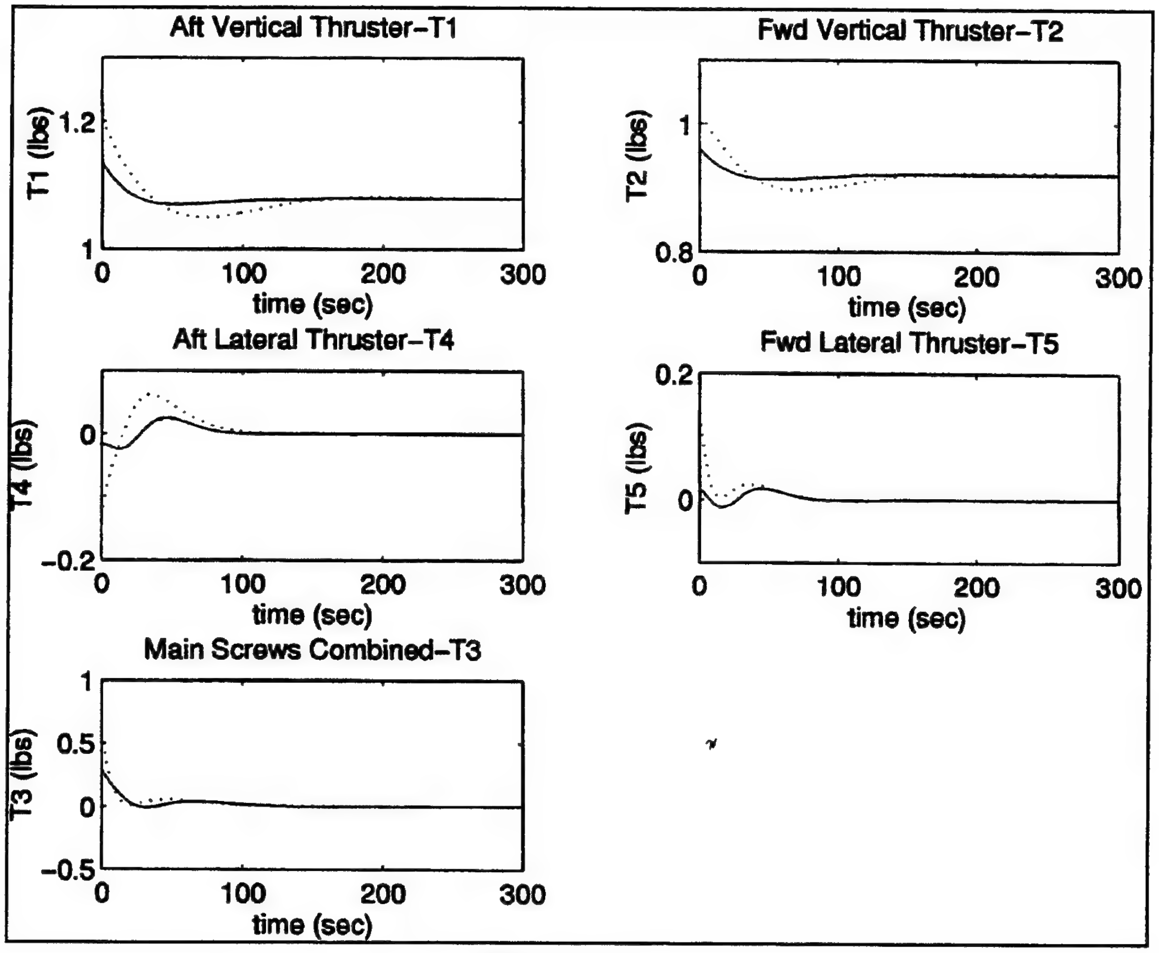

Figure 36

MATLAB Design-1 (Solid) vs. LISP Design-2 (Dotted) Thruster Force Results Run #2

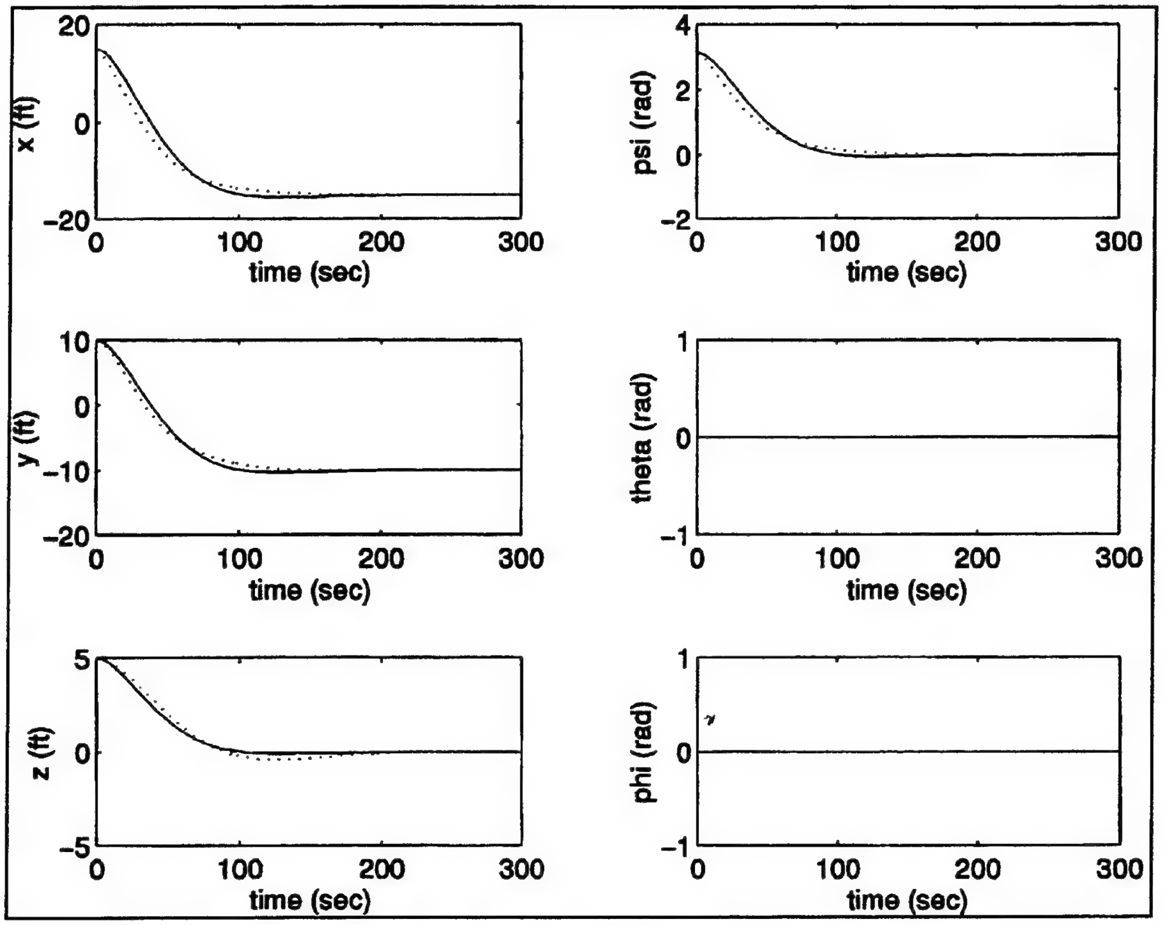

Figure 37

MATLAB Design-1 (Solid) vs. LISP Design-2 (Dotted) Posture Results Run #3

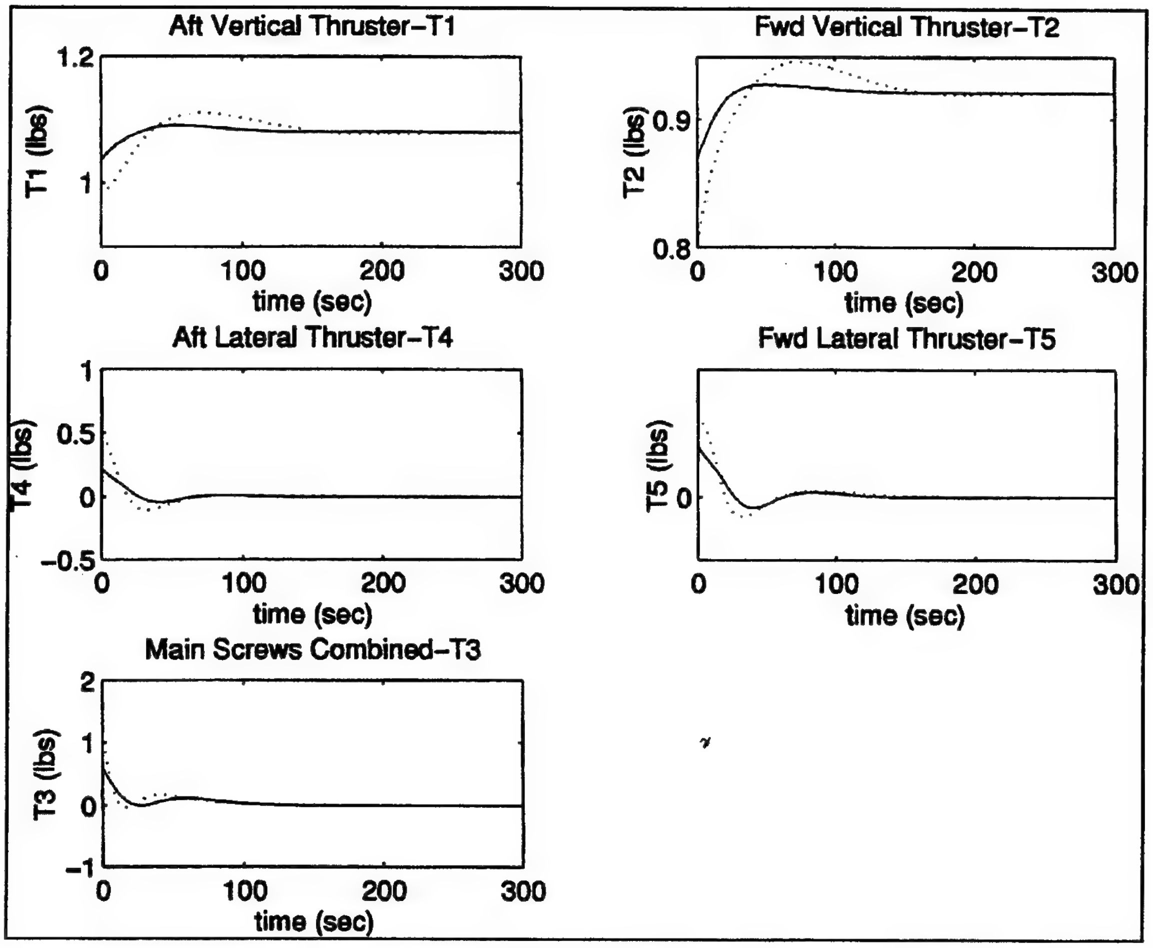

Figure 38

MATLAB Design-1 (Solid) vs. LISP Design-2 (Dotted) Thruster Force Results Run #3

VII. SUMMARY

A. SYNOPSIS

This thesis presented a basic design for a hover mode autopilot to possibly be incorporated into the NPS Phoenix AUV. The autopilot was implemented and simulated using Common LISP object oriented programming language. Initially, the autopilot design was developed within the constraints of the initial design boundaries. This first design produced favorable results in controlling five of the six degrees of freedom. After successfully implementing and simulating the initial design an additional factor of reality was added. This additional factor was the diagonal terms of the added mass effects. In the case of the second design, favorable results were obtained when controlling surge, sway, heave, and yaw. When an attempt to include the control of pitch was made, the autopilot produced erroneous results. However, this problem is assumed to be caused by an error in the LISP source code. This assumption is based on the fact that a MATLAB implementation of the modified autopilot design provided meaningful results. Although this problem has not been solved, the ability to control the four degrees of freedom mentioned above simultaneously is a significant improvement over the current hover mode autopilot implemented in the vehicle where these actions are performed separately.

B. RECOMMENDATIONS FOR FURTHER STUDY

This thesis investigated the design and simulation of a hover mode autopilot using LISP. The control design proved to be favorable with some exceptions. As a result, further investigation into the control design presented is possible.

The fact that the second controller design may not have been properly implemented with the LISP code provides the opportunity for a detailed investigation of the source code. Knowledge in programming with LISP as well as an adequate amount control theory is an essential requirement to try and correct the problem with controlling pitch.

Further investigation of the autopilot controller design is also a potential source of study. The controller design in and of itself is substantial. Expanding Chapter IV to eliminate many of the boundaries as set forth in Chapter III is a worthy course of action. The reason for this action is that it may provide a more realistic model which would produce simulated results that are closer to the actual vehicle responses.

APPENDIX A. LISP SOURCE CODE

This appendix contains the all the LISP code for creating the Phoenix AUV polygon model for which the autopilot design is applied.

A. OPERATING INSTRUCTIONS

In order to run the simulation the following instructions are provided. They assume that the required files are installed in the current directory and the user has access to and a basic familiarity of using Common LISP.

- At the prompt call up LISP.

- Load the required files by typing (load "load-fasl").

- Initialize the simulation by typing (initialize-all). This will set all parameters to default values. If different values are desired, one only need to understand the file "initialization.cl".

- Run the simulation by typing (autopilot dt stop-time), where dt is the desired time step and stop-time is the total run time in seconds. Note that using 0.01 for dt provides the best results.

After following the above instructions two windows will open and provide an overhead view as well as a view side view. The numerical results for the posture values will be displayed in the window from which LISP was started and they will also be routed to a file named "auvsim.dat" for later analysis.

B. AUV-RIGID-BODY.CL

This file was originally written by Dr. McGhee to create a generic rigid-body. It has been modified by the author to create the Phoenix AUV rigid-body.

```
(defclass rigid-body
  ()
 ((posture              ;The six-vector (psi theta phi x y z) in world coordinates.
  :initform '(0 0 0 0 0 0)
  :initarg :posture
  :accessor posture)
 (velocity              ;The six-vector (u v w p q r) in body coordinates.
  :initform '(0 0 0 0 0 0)
  :initarg :velocity
  :accessor velocity)
 (velocity-growth-rate  ;The vector (u-dot v-dot w-dot p-dot q-dot r-dot).
  :initform '(0 0 0 0 0 0)
  :accessor velocity-growth-rate)
 (forces-and-torques    ;The vector (Fx Fy Fz L M N) in body coordinates.
  :initform '(0 0 0 0 0 0)
  :accessor forces-and-torques)
 (moments-of-inertia    ;The vector (Ix Iy Iz) in principal axis coordinates.
  :initform '(1 1 1)
  :initarg :moments-of-inertia
  :accessor moments-of-inertia)
 (mass
  :initform '(1 1 1)
  :initarg :mass
  :accessor mass)
 (node-list    ;(x y z 1) in body coord for each node. Starts with (0 0 0 1).
  :initform '((0 0 0 1) (4 0 0 1) (2 0 0 1) (-4 0 0 1) (-5 0 -2 1)
            (-6 -1.5 -2 1) (-6 1.5 -2 1) (-2 6 -2 1) (-2 -6 -2 1))
  :initarg :node-list
  :accessor node-list)
 (polygon-list
  :initform '((1 2 3 4 5) (4 6) (7 2 8))
  :initarg :polygon-list
  :accessor polygon-list)
 (transformed-node-list  ;(x y z 1) in earth coord for each node in node-list.
  :accessor transformed-node-list)
 (H-matrix
  :initform (unit-matrix 4)
  :accessor H-matrix)
 (delta-t
  :accessor delta-t)))
```

```
;; define AUV-body subclass of rigid-body
(defclass AUV-body (rigid-body)

        ((node-list
        :initform '((0 0 0 1) (-3.58 -.67 0 1) (-2.25 -.67 -.42 1) (2.54 -.67 -.42 1)
                (2.75 -.63 -.4 1) (3 -.58 -.35 1) (3.17 -.5 -.29 1) (3.38 -.38 -.21 1)
                (3.58 -.21 -.08 1) (3.58 -.21 .08 1) (3.38 -.38 .21 1) (3.17 -.5 .29 1)
                (3 -.58 .34 1) (2.75 -.63 .4 1) (2.54 -.67 .42 1) (-2.25 -.67 .42 1)
                (-3.58 .67 0 1) (-2.25 .67 -.42 1) (2.54 .67 -.42 1) (2.75 .63 -.4 1)
                (3 .58 -.35 1) (3.17 .5 -.29 1) (3.38 .38 -.21 1) (3.58 .21 -.08 1)
                (3.58 .21 .08 1) (3.38 .38 .21 1) (3.17 .5 .29 1) (3 .58 .34 1)
                (2.75 .63 .4 1) (2.54 .67 .42 1) (-2.25 .67 .42 1)
                (-2.67 0 -.42 1) (-2.46 0 -.84 1) (-2.25 0 -.84 1) (-2.25 0 -.42 1)
                (-2.67 0 .42 1) (-2.46 0 .84 1) (-2.25 0 .84 1) (-2.25 0 .42 1)
                (-2.67 -.67 0 1) (-2.46 -1.09 0 1) (-2.25 -1.09 0 1) (-2.25 -.67 0 1)
                (-2.67 .67 0 1) (-2.46 1.09 0 1) (-2.25 1.09 0 1) (-2.25 .67 0 1)
                (2.54 0 -.42 1) (2.54 0 -.84 1) (2.33 0 -.84 1) (2.12 0 -.42 1)
                (2.54 0 .42 1) (2.54 0 .84 1) (2.33 0 .84 1) (2.12 0 .42 1)
                (2.54 -.67 0 1) (2.54 -1.09 0 1) (2.33 -1.09 0 1) (2.12 -.67 0 1)
                (2.54 .67 0 1) (2.54 1.09 0 1) (2.33 1.09 0 1) (2.12 .67 0 1)))
        (polygon-list
        :initform '((1 2 3 4 5 6 7 8 9 10 11 12 13 14 15)
                (16 17 18 19 20 21 22 23 24 25 26 27 28 29 30) (8 9 23 24) (1 16)
                ; (2 17) (3 18) (14 29) (15 30)
                 (31 32 33 34) (35 36 37 38) (39 40 41 42)
                (43 44 45 46) (47 48 49 50) (51 52 53 54) (55 56 57 58) (59 60 61 62)))
        (thrust-com                     ;; thrusters T1 T2 T3 T4 T5
        :initform '(0 0 0 0 0)
        :initarg :thrust-com
        :accessor thrust-com)
        (gains
        ;; Kx Kxdot Ky Kydot Kz Kzdot Kpsi Kpsidot Ktheta  Kthetadot Kphi Kphidot
        :initform '(1 1 1 1 1 1 1 1 1 1 1 1)
        :initarg :gains
        :accessor gains)
        (desired-posture                ;; xo yo zo psio thetao phio
        :initform '(0 0 0 0 0 0)
        :initarg :desired-posture
        :accessor desired-posture)
        (posture-rates                  ;; xdot ydot zdot psidot thetadot phidot
        :initform '(0 0 0 0 0 0)
        :initarg :posture-rates
        :accessor posture-rates)
        (forces-earth-coord                     ;; Fxe Fye Fze Tpsi Ttheta Tphi
        :initform '(0 0 0 0 0 0)
        :initarg :forces-earth-coord
        :accessor forces-earth-coord)
        (physical-char                  ;; m I Fb Fw
        :initform '(1 1 1 1)
```

```
          :initarg :physical-char
          :accessor physical-char)
          (R-matrix
          :initform (unit-matrix 3)
          :accessor R-matrix)
       (added-m-coeff
       :initform '(0 0 0)
       :accessor added-m-coeff)
       (added-I-coeff
       :initform '(0 0 0)
       :accessor added-I-coeff)
)) ;close AUV-body subclass
```

C. B2E-TRANSFORMATIONS.CL

This file was written by the author to transform body translational and rotational velocities to earth translational velocities and Euler angle rates.

```
;; function to solve for psi-dot, theta-dot, phi-dot  (Euler angle rates)
(defmethod solve-euler-angle-rates ((body rigid-body))
    (multiple-value-bind (u v w p q r psi theta phi)
          (values-list (append (velocity body) (firstn 3 (posture body))))
     (list
       (+ (* (sec theta) (sin phi) q)
          (* (sec theta) (cos phi) r))
       (-  (* (cos phi) q)
          (* (sin phi) r))
       (+ p
          (* (tan theta) (sin phi) q)
          (* (tan theta) (cos phi) r)))))

;; function to solve for x-dot, y-dot, z-dot in earth coordinates
(defmethod solve-xyz-dot-earth ((body rigid-body))
 (post-multiply (R-matrix body) (firstn 3 (velocity body))))
```

D. INITIALIZATION.CL

This code was written by the author to initialize the simulation by setting the parameters and default initial conditions.

```
;; function to initialize everything to default values
(defmethod initialize-all ()
  (setf sub (make-instance 'AUV-body))
   (set-parameters 12.0117 1.0650 12.9541 35.2744 2.7 42 45
                                4.8330 125.8594 9.4646 389 .04 .8 sub)
  (set-init-cond 0 0 0 0 0 0 0 0 0 0 0 0 sub)
  (set-desired-cond 15 10 5 pi 0 0 sub)
  (init-cameras)
); close initialize-all

;; function to reset to default values and clear existing cameras
(defmethod reset ()
  (setf sub (make-instance 'AUV-body))
    (set-parameters 12.0117 1.0650 12.9541 35.2744 2.7 42 45
                                4.8330 125.8594 9.4646 389 .04 .8 sub)
  (set-init-cond 0 0 0 0 0 0 0 0 0 0 0 0 sub)
  (set-desired-cond 15 10 5 pi 0 0 sub)
  (clear-cameras)
); close reset


;; function to set physical parameters (mass, inertia, etc.)
(defmethod set-parameters (mass xudot yvdot zwdot Ix Iy Iz
                                kpdot mqdot nrdot Fb w z (body rigid-body))
    (setf (added-m-coeff body) (list xudot yvdot zwdot))
    (setf (added-I-coeff body) (list kpdot mqdot nrdot))
    (setf (moments-of-inertia body)
            (vector-add (list Ix Iy Iz) (added-I-coeff body)))
    (setf (mass body)
            (vector-add (list mass mass mass) (added-m-coeff body)))
    (setf m (average-vector(vector-add (list mass mass mass) (added-m-coeff body)) ))
    (setf I (average-vector (cdr (vector-add (list Ix Iy Iz) (added-I-coeff body)) )))

    (setf (forces-and-torques body) (list 0 0 (- (* *gravity* (+ zwdot mass))) 0 0 0))

    (setf (physical-char body) (list m I (+ (* *gravity* zwdot) Fb) (* *gravity* (+ zwdot mass))))


    (setf (gains body)
            (list  (* m (expt w 2))            ;Kx
                   (* 2 z w m)                ;Kxdot
```

```
                (* m (expt w 2))            ;Ky
                (* 2 z w m)                 ;Kydot
                (* m (expt w 2))            ;Kz
                (* 2 z w m)                 ;Kzdot
                (* I (expt w 2))            ;Kpsi
                (* 2 z w I)                 ;Kpsidot
                (* I (expt w 2))            ;Ktheta
                (* 2 z w I)                 ;Kthetadot
                (* I (expt w 2))            ;Kphi
                (* 2 z w I) ))              ;Kphidot

);close set-parameters


;; function to set initial conditions for AUV
(defmethod set-init-cond
  (x xdot y ydot z zdot psi psidot theta thetadot phi phidot (body rigid-body))
  (setf (posture body) (list  psi theta phi x y z))
  (setf (posture-rates body) (list psidot thetadot phidot xdot ydot zdot ))
);close set-init-cond


;; function to input desired condtions (position, angle etc.) for AUV
(defmethod set-desired-cond (xo yo zo psio thetao phio (body rigid-body))
     (setf (desired-posture body) (list xo yo zo psio thetao phio))
); close set-desired-cond


;;defmethod to initialize required cameras
(defmethod init-cameras ()
  (setf camera-3 (make-instance 'strobe-camera))
  (setf camera-2 (make-instance 'strobe-camera))
  (move camera-3 0 (/ pi -2) 0 0 0 -25)
  (move camera-2 (- (/ pi 2 )) 0 0 0 25 0)
);close init-cameras


;;defmethod to take a round of pictures
(defmethod grp-picture ()
              (take-picture camera-2 sub)
              (take-picture camera-3 sub)
);close grp-picture
```

E. MISC-FUNCTIONS.CL

This file contains functions that the author found necessary to create.

```
;;Misc functions

(defun eleventh (l) (nth 10 l))

(defun twelfth (l) (nth 11 l))

(defconstant *gravity* 32.2185)

(defun sec (angle) (/ 1 (cos angle)))


(defmethod clear-cameras ()
 (cw:clear (camera-window camera-2))
 (cw:clear (camera-window camera-3))
); close clear-cameras

;; function to display posture values on screen during execution
(defmethod show-values ((body rigid-body))
    (setf x (posture body))
    (format t  "~% ~6,3f ~6,3f ~6,3f ~6,3f ~6,3f ~6,3f "
       (first x) (second x) (third x) (fourth x) (fifth x) (sixth x))
); close show-values

 (defun average-vector (vector)
      (setf sum 0 count 0)
      (dolist (element vector average)
          (setf sum (+ sum element))
          (setf count (+ count 1))
          (setf average (/ sum count))))
```

F. ROBOT-KINEMATICS.CL

This is original code was written by Dr. McGhee. It contains the necessary functions associated for kinematic modeling.

```
(defun transpose (matrix)          ;A matrix is a list of row vectors.
  (cond ((null (cdr matrix)) (mapcar 'list (car matrix)))
        (t (mapcar 'cons (car matrix) (transpose (cdr matrix))))))

(defun dot-product (vector-1 vector-2)  ;A vector is a list of numerical atoms.
  (apply '+ (mapcar '* vector-1 vector-2)))

(defun vector-magnitude (vector) (sqrt (dot-product vector vector)))

(defun post-multiply (matrix vector)
  (cond ((null (rest matrix)) (list (dot-product (first matrix) vector)))
        (t (cons (dot-product (first matrix) vector)
             (post-multiply (rest matrix) vector)))))

(defun pre-multiply (vector matrix)
  (post-multiply (transpose matrix) vector))

(defun matrix-multiply (A B)      ;A and B are conformable matrices.
  (cond ((null (cdr A)) (list (pre-multiply (car A) B)))
        (t (cons (pre-multiply (car A) B) (matrix-multiply (cdr A) B)))))

(defun chain-multiply (L)        ;L is a list of names of conformable matrices.
  (cond ((null (cddr L)) (matrix-multiply (eval (car L)) (eval (cadr L))))
        (t (matrix-multiply (eval (car L)) (chain-multiply (cdr L))))))

(defun cycle-left (matrix) (mapcar 'row-cycle-left matrix))

(defun row-cycle-left (row) (append (cdr row) (list (car row))))

(defun cycle-up (matrix) (append (cdr matrix) (list (car matrix))))

(defun unit-vector (one-column length)        ;Column count starts at 1.
  (do ((n length (1- n))
       (vector nil (cons (cond ((= one-column n) 1) (t 0)) vector)))
      ((zerop n) vector)))

(defun unit-matrix (size)
  (do ((row-number size (1- row-number))
       (I nil (cons (unit-vector row-number size) I)))
      ((zerop row-number) I)))
```

```
(defun concat-matrix (A B)   ;A and B are matrices with equal number of rows.
  (cond ((null A) B)
           (t (cons (append (car A) (car B)) (concat-matrix (cdr A) (cdr B))))))

(defun augment (matrix)
  (concat-matrix matrix (unit-matrix (length matrix))))

(defun normalize-row (row) (scalar-multiply (/ 1.0 (car row)) row))

(defun scalar-multiply (scalar vector)
  (cond ((null vector) nil)
           (t (cons (* scalar (car vector))
                (scalar-multiply scalar (cdr vector))))))

(defun solve-first-column (matrix)        ;Reduces first column to (1 0 ... 0).
  (do* ((remaining-row-list matrix (rest remaining-row-list))
        (first-row (normalize-row (first matrix)))
        (answer (list first-row)
              (cons (vector-add (first remaining-row-list)
                             (scalar-multiply (- (caar remaining-row-list))
                                         first-row))
                 answer)))
       ((null (rest remaining-row-list)) (reverse answer))))

(defun vector-add (vector-1 vector-2) (mapcar '+ vector-1 vector-2))

(defun vector-subtract (vector-1 vector-2) (mapcar '- vector-1 vector-2))

(defun first-square (matrix)  ;Returns leftmost square matrix from argument.
  (do ((size (length matrix))
       (remainder matrix (rest remainder))
       (answer nil (cons (firstn size (first remainder)) answer)))
      ((null remainder) (reverse answer))))

(defun firstn (n list)
  (cond ((zerop n) nil)
        (t (cons (first list) (firstn (1- n) (rest list))))))

(defun max-car-firstn (n list)
  (append (max-car-first (firstn n list)) (nthcdr n list)))

(defun matrix-inverse (M)
  (do ((M1 (max-car-first (augment M))
         (cond ((null M1) nil)
                  (t (max-car-firstn n (cycle-left (cycle-up M1))))))
       (n (1- (length M)) (1- n)))
      ((or (minusp n) (null M1)) (cond ((null M1) nil) (t (first-square M1))))
      (setq M1 (cond ((zerop (caar M1)) nil) (t (solve-first-column M1))))))
```

```
(defun max-car-first (L)   ;L is a list of lists. This function finds list with
  (cond ((null (cdr L)) L) ;largest car and moves it to head of list of lists.
      (t (if (> (abs (caar L)) (abs (caar (max-car-first (cdr L))))) L
            (append (max-car-first (cdr L)) (list (car L)))))))

(defun dh-matrix (cosrotate sinrotate costwist sintwist length translate)
  (list (list cosrotate (- (* costwist sinrotate))
                (* sintwist sinrotate) (* length cosrotate))
          (list sinrotate (* costwist cosrotate)
                (- (* sintwist cosrotate)) (* length sinrotate))
          (list 0. sintwist costwist translate) (list 0. 0. 0. 1.)))

(defun homogeneous-transform (azimuth elevation roll x y z)
  (rotation-and-translation (sin azimuth) (cos azimuth) (sin elevation)
  (cos elevation) (sin roll) (cos roll) x y z))

(defun rotation-and-translation (spsi cpsi sth cth sphi cphi x y z)
  (list (list (* cpsi cth) (- (* cpsi sth sphi) (* spsi cphi))
                (+ (* cpsi sth cphi) (* spsi sphi)) x)
          (list (* spsi cth) (+ (* cpsi cphi) (* spsi sth sphi))
                (- (* spsi sth cphi) (* cpsi sphi)) y)
          (list (- sth) (* cth sphi) (* cth cphi) z)
          (list 0. 0. 0. 1.)))

(defun inverse-H (H)          ;H is a 4x4 homogeneous transformation matrix.
  (let* ((minus-P (list (- (fourth (first H)))
                    (- (fourth (second H)))
                    (- (fourth (third H)))))
       (inverse-R (transpose (first-square (reverse (rest (reverse H))))))
       (inverse-P (post-multiply inverse-R minus-P)))
      (append (concat-matrix inverse-R (transpose (list inverse-P)))
            (list (list 0 0 0 1)))))

(setf x '((1 2) (3 4)))
```

G. RUN-SIMULATION.CL

This code was written by the author. It is the top layer in the simulation cycle.

```
;; AUV autopilot simulation using euler solution to linear equations
(defmethod autopilot (time-step halt-time)
  (setf (delta-t sub) time-step)
  (update-rigid-body sub)

  (grp-picture)

(with-open-file (output-stream "auvsim.dat" :direction :output)
 (do (
         (time 0 (+ time time-step))
         (sec-count 0 (+ sec-count time-step))
         (sec-count2 0 (+ sec-count2 time-step)))
         ((> time halt-time) 'done)
         (linear-feedback sub)
         (update-thrusters sub)
         (check-thruster-limit sub)
         (update-forces-and-torques sub)
         (update-rigid-body sub)
    (when (> sec-count 10)
        (grp-picture)
            (setf sec-count 0)
     ); close when
         (show-values sub)
         (when (> = sec-count2 1)
         (setf x (posture sub))
         (setf f (thrust-com sub))
         (setf v (posture-rates sub))
         (format output-stream "~ % ~6,3f ~6,3f ~6,3f ~6,3f ~6,3f ~6,3f  ~6,3f ~6,3f ~6,3f ~6,3f
~6,3f ~6,3f ~6,3f ~6,3f ~6,3f ~6,3f ~6,3f"
                  (first x) (second x) (third x) (fourth x) (fifth x) (sixth x)
                  (first v) (second v) (third v) (fourth v) (fifth v) (sixth v)
                  (first f) (second f) (third f) (fourth f) (fifth f))
         (setf sec-count2 0)
         ); close when
 );close do
 );close with-open auvsim.dat
);close autopilot
```

H. UPDATE-FUNCTIONS.CL

This code contains functions written by Dr. McGhee and the author. It contains all the functions described in Chapter V.

```
(defmethod update-rigid-body ((body rigid-body))     ;Euler integration.
  (let ((delta-t (delta-t body)))
          (update-posture body)
          (update-velocity body delta-t)
          (update-velocity-growth-rate body)
          (update-posture-rates body))
);close update-rigid-body

;; function using Newton-Euler equations
(defmethod update-velocity-growth-rate ((body rigid-body))
  (setf (velocity-growth-rate body) ;Assumes principal axis coordinates with
    (multiple-value-bind            ;origin at center of gravity of body.
      (Fx Fy Fz L M N  u v w p q r  Ix Iy Iz mx my mz) ;Declares local variables.
      (values-list                         ;Values assigned.
        (append (forces-and-torques body) (velocity body) (moments-of-inertia body) (mass body)))
        (list (+ (* v r) (* -1 w q) (/ Fx mx)
               (* *gravity* (first (third (H-matrix body)))))
             (+ (* w p) (* -1 u r) (/ Fy my)
                (* *gravity* (second (third (H-matrix body)))))
             (+ (* u q) (* -1 v p) (/ Fz mz)
                (* *gravity* (third (third (H-matrix body)))))
                (/ (+ (* (- Iy Iz) q r) L) Ix)
                (/ (+ (* (- Iz Ix) r p) M) Iy)
                (/ (+ (* (- Ix Iy) p q) N) Iz)))) ;Value returned.
); close update-velocity-growth-rate

(defmethod update-velocity ((body rigid-body) delta-t)
  (setf (velocity body)
    (vector-add (velocity body)
                    (scalar-multiply delta-t (velocity-growth-rate body))))
); close update-velocity

(defmethod update-posture ((body rigid-body))
    (move body
      (first (posture body))
      (second (posture body))
      (third (posture body))
      (fourth (posture body))
      (fifth (posture body))
      (sixth (posture body)))
); close update-posture
```

```
(defmethod move ((body rigid-body) azimuth elevation roll x y z)
  (setf (H-matrix body)
        (homogeneous-transform azimuth elevation roll x y z))
  (setf (R-matrix body) (extract-R-matrix (H-matrix body)))
  (transform-node-list body)
  (update-position body)
); close move


(defmethod transform-node-list ((body rigid-body))
     (setf (transformed-node-list body)
           (mapcar #'(lambda (node-location)
                     (post-multiply (H-matrix body) node-location))
              (node-list body))))

(defmethod update-position ((body rigid-body))
  (setf (posture body) (vector-add (posture body) (scalar-multiply (delta-t body)
          (posture-rates body))))
); close update-position


(defmethod update-posture-rates ((body rigid-body))
        (setf (posture-rates body)
               (append (solve-euler-angle-rates body) (solve-xyz-dot-earth body)))
);close update-posture-rates


;;function to determine Fxe, Fye, Fze, Tphi, Ttheta, Tpsi
(defmethod linear-feedback ((body rigid-body))
  (setf (forces-earth-coord body)
    (multiple-value-bind
          (psi theta phi x y z xo yo zo psio thetao phio Kx Kxdot Ky Kydot Kz Kzdot
          Kpsi Kpsidot Ktheta Kthetadot Kphi Kphidot psidot thetadot phidot xdot ydot
          zdot Fb Fw)
          (values-list (append (posture body) (desired-posture body) (gains body)
                    (posture-rates body) (nthcdr 2 (physical-char body))))
          (list
          (- (+ (* Kx (- x xo)) (* Kxdot xdot)))
          (- (+ (* Ky (- y yo)) (* Kydot ydot)))
          (- Fb Fw (+ (* Kz (- z zo)) (* Kzdot zdot)))
          0
          (- (* Fb .04 (sin theta)) (+ (* Ktheta (- theta thetao)) (* Kthetadot thetadot)))
          (- (+ (* Kpsi (- psi psio)) (* Kpsidot psidot))))))
); close linear-feedback

;;function to extract the R-matrix from the H-matrix
(defmethod extract-R-matrix (H-matrix)
  (list (firstn 3 (first H-matrix))
          (firstn 3 (second H-matrix))
          (firstn 3 (third H-matrix)))
); close extract-R-matrix

;; function to update forces and moments in body coordinates
(defmethod update-forces-and-torques ((body rigid-body))
  (setf (forces-and-torques body)
          (multiple-value-bind
           (T1 T2 T3 T4 T5 Fb psi theta phi Kphidot phidot u v w p q r)
           (values-list (append (thrust-com body) (list (third (physical-char body)))
           (firstn 3 (posture body)) (list (twelfth (gains body)) (third (posture-rates body)))
           (velocity body)))
                    (list
                     (+ T3 (* Fb (sin theta)) )
                     (+ T4 T5 (- (* Fb (cos theta) (sin phi))))
                     (+ T1 T2 (- (* Fb (cos theta) (cos phi))) )
                     (- (* T5 .04) (* T4 .04) (* Kphidot p)(* Fb .04 (sin phi)))
                     (- (* T1 1.23) (* T2 1.44) (* T3 .04)  (* Fb .04 (sin theta)))
                     (- (* T5 1.94) (* T4 1.73)))))
); close update-forces-and-torques

;; Thruster force calculations
(defmethod update-thrusters ((body rigid-body))
  (setf thrust-coeff-matrix '((0 0 1 0 0) (0 0 0 1 1) (1 1 0 0 0) (1.23 -1.44 -.04 0 0)
                                                    (0 0 0 -1.73 1.94)))
  (setf A (append
            (post-multiply (transpose (R-matrix body)) (firstn 3 (forces-earth-coord body)))
         (nthcdr 4 (forces-earth-coord body))))
  (setf (thrust-com body) (post-multiply (matrix-inverse thrust-coeff-matrix) A))
) ;close update-thrusters

;; function to check if thrusters are within limits
(defmethod check-thruster-limit ((body rigid-body))
  (setf T3 (third (thrust-com body)))
  (if (> (abs T3) 10)
    (setf T3 (* 10 (/ T3 (abs T3)))))
  (setf T1245 (append (firstn 2 (thrust-com body)) (nthcdr 3 (thrust-com body))))
  (setf final-t nil)
  (setf T1245
          (dolist (thrust T1245 final-t)
                     (if (> (abs thrust) 2)
                       (setf final-t (append final-t (list (* 2 (/ thrust (abs thrust))))))
            (setf final-t (append final-t (list thrust))))))
  (setf (thrust-com body) (append (firstn 2 T1245) (list T3) (nthcdr 2 T1245)))
); close check-thruster-limit
```

I. STROBE-CAMERA.CL

This code was written by Dr. McGhee. It allows the graphically simulation of the vehicle.

```
(require :xcw)
(cw:initialize-common-windows)
(defclass strobe-camera (rigid-body)
 ((focal-length
   :accessor focal-length
   :initform 6)
  (camera-window
   :accessor camera-window
   :initform (cw:make-window-stream :borders 5
                                    :left 10
                                    :bottom 10
                                    :width 600
                                    :height 600
                                    :title "strobe-camera-image"
                                    :activate-p t))
  (H-matrix
   :initform (homogeneous-transform .3 -.3 0 -300 -90 -90))
  (inverse-H-matrix
   :accessor inverse-H-matrix
   :initform (inverse-H (homogeneous-transform .3 -.3 0 -300 -90 -90)))
  (enlargement-factor
   :accessor enlargement-factor
   :initform 30)))
(defmethod move ((camera strobe-camera) azimuth elevation roll x y z)
 (setf (H-matrix camera) (homogeneous-transform azimuth elevation roll x y z))
 (setf (inverse-H-matrix camera) (inverse-H (H-matrix camera))))
(defmethod take-picture ((camera strobe-camera) (body rigid-body))
 (let ((camera-space-node-list (mapcar #'(lambda (node-location)
       (post-multiply (inverse-H-matrix camera) node-location))
               (transformed-node-list body))))
   (dolist (polygon (polygon-list body))
     (clip-and-draw-polygon camera polygon camera-space-node-list))))

(defmethod clip-and-draw-polygon
 ((camera strobe-camera) polygon node-coord-list)
 (do* ((initial-point (nth (first polygon) node-coord-list))
       (from-point initial-point to-point)
       (remaining-nodes (rest polygon) (rest remaining-nodes))
       (to-point (nth (first remaining-nodes) node-coord-list)
             (if (not (null (first remaining-nodes)))
                 (nth (first remaining-nodes) node-coord-list))))
      ((null to-point)
          (draw-clipped-projection camera from-point initial-point))
      (draw-clipped-projection camera from-point to-point)))

(defmethod draw-clipped-projection ((camera strobe-camera) from-point to-point)
  (cond ((and (<= (first from-point) (focal-length camera))
              (<= (first to-point) (focal-length camera))) nil)
        ((<= (first from-point) (focal-length camera))
         (draw-line-in-camera-window camera
           (perspective-transform camera (from-clip camera from-point to-point))
           (perspective-transform camera to-point)))
        ((<= (first to-point) (focal-length camera))
         (draw-line-in-camera-window camera
           (perspective-transform camera from-point)
           (perspective-transform camera (to-clip camera from-point to-point))))
        (t (draw-line-in-camera-window camera
           (perspective-transform camera from-point)
           (perspective-transform camera to-point)))))

(defmethod from-clip ((camera strobe-camera) from-point to-point)
  (let ((scale-factor (/ (- (focal-length camera) (first from-point))
                         (- (first to-point) (first from-point)))))
      (list (+ (first from-point)
               (* scale-factor (- (first to-point) (first from-point))))
            (+ (second from-point)
               (* scale-factor (- (second to-point) (second from-point))))
            (+ (third from-point)
               (* scale-factor (- (third to-point) (third from-point)))) 1)))

(defmethod to-clip ((camera strobe-camera) from-point to-point)
  (from-clip camera to-point from-point))

(defmethod draw-line-in-camera-window ((camera strobe-camera) start end)
  (cw:draw-line (camera-window camera)
              (cw:make-position :x (first start) :y (second start))
              (cw:make-position :x (first end) :y (second end))
              :brush-width 0))

(defmethod perspective-transform ((camera strobe-camera) point-in-camera-space)
  (let* ((enlargement-factor (enlargement-factor camera))
         (focal-length (focal-length camera))
         (x (first point-in-camera-space))  ;x axis is along optical axis
         (y (second point-in-camera-space)) ;y is out right side of camera
         (z (third point-in-camera-space))) ;z is out bottom of camera
        (list (+ (round (* enlargement-factor (/ (* focal-length y) x)))
                   300)                         ;to right in camera window
         (+ 300 (round (* enlargement-factor (/ (* focal-length (- z)) x))
                   ))))) ;up in camera window
```

APPENDIX B. MATLAB SOURCE CODE

```
% Simulation of Thruster Forces for AUV in 3-D

% Define global variables
global m I g xo yo zo psio thetao phio omega zeta K Kdot mam Ixx Iyy Izz mwam
global Kx Kxdot Ky Kydot Kz Kzdot Kpsi Kpsidot Ktheta Kthetadot Kphi Kphidot

d1=1.23;                % longitudinal distance between cg and T1 (thuster-1)
d2=1.44;                % longitudinal distance between cg and T2
d3=0.04;                % vertical distance between cg and T3p/T3s
d4=1.73;                % longitudinal distance between cg and T5
d5=1.94;                % longitudinal distance between cg and T6
                        % all distances in feet

% initial conditions
icq=input('use default initial conditions? ','s');
if icq=='y'
x=0;                    xdot=0;
y=0;                    ydot=0;
z=0;                    zdot=0;
psi=0;                  psidot=0;
theta=0;                thetadot=0;
phi=0;                  phidot=0;
else
ic=input('set initial posture and rates = ');
x=ic(1);          xdot=ic(2);
y=ic(3);          ydot=ic(4);
z=ic(5);          zdot=ic(6);
psi=ic(7);        psidot=ic(8);
theta=ic(9);      thetadot=ic(10);
phi=ic(11);       phidot=ic(12);
end

%desired conditions
dc=input('set desired conditions = ');
xo=dc(1);               yo=dc(2);               zo=dc(3);
psio=dc(4);             thetao=dc(5);           phio=dc(6);

% AUV parameters
Fb=389;                 % buouancy force in lbs.
g=32.2185;              % gravity
m=12.0117;              % mass in slugs (dry weight asssumed to be 387 lbs.)
I=12.0117;              % inertia
Ixx=2.7;                      Iyy=42;           Izz=45;
kpdot=4.833;            mqdot=125.8594;         nrdot=9.4646;
xudot=1.0650;           yvdot=12.9541;          zwdot=35.2744;
```

```
mwam=[(Ixx+kpdot) (Iyy+mqdot) (Izz+nrdot) (m+xudot) (m+yvdot) (m+zwdot)];
aI=((Iyy+mqdot)+ (Izz+nrdot))/2;
am=((m+xudot)+ (m+yvdot)+ (m+zwdot))/3;
mam=[aI aI aI am am am];

% omega and zeta values
omega=[.04 .04 .04 .04 .04 .04];   % omega values for psi,theta,phi,x,y,z respectively
zeta= [.8 .8 .8 .8 .8 .8];         % zeta values for psi,theta,phi,x,y,z respectively

% Gains
K=mam.*(omega.^2);
Kdot=2.*mam.*zeta.*omega;

Kpsi=K(1);                Kpsidot=Kdot(1);
Ktheta=K(2);              Kthetadot=Kdot(2);
Kphi=K(3);                Kphidot=Kdot(3);
Kx=K(4);                  Kxdot=Kdot(4);
Ky=K(5);                  Kydot=Kdot(5);
Kz=K(6);                  Kzdot=Kdot(6);

% start and stop times
t0=0;
tfinal=input('set tfinal = ');

% Numerical Integration
v0=[x xdot z zdot theta thetadot psi psidot phi phidot y ydot]';
[t,v]=ode23('vd3d',t0,tfinal,v0);

x=v(:,1); xdot=v(:,2); z=v(:,3); zdot=v(:,4); theta=v(:,5); thetadot=v(:,6);
psi=v(:,7); psidot=v(:,8); phi=v(:,9); phidot=v(:,10); y=v(:,11); ydot=v(:,12);

% Matrix of Coefficients for T1,T2,T3p,T3s,T5,T6 and its inverse matrix
c=[0 0 1 0 0;0 0 0 1 1;1 1 0 0 0;d1 -d2 -d3 0 0;0 0 0 -d4 d5];
cinv=inv(c);

% Control Equations
Fxe=-Kx.*(x-xo) - Kxdot.*xdot;
Fye=-Ky.*(y-yo) - Kydot.*ydot;
Fze=-Kz.*(z-zo) - Kzdot.*zdot + (Fb-m*g);
Fe=[Fxe Fye Fze]';

% Euler Transformation Matrix
e11=cos(psi).*cos(theta);
e12=(cos(psi).*sin(theta).*sin(phi) - sin(psi).*cos(phi));
e13=(cos(psi).*sin(theta).*cos(phi) + sin(psi).*sin(phi));
e21=sin(psi).*cos(theta);
e22=(cos(psi).*cos(phi) + sin(psi).*sin(theta).*sin(phi));
e23=(-cos(psi).*sin(phi) + sin(psi).*sin(theta).*cos(phi));
e31=-sin(theta);
```

```
e32=cos(theta).*sin(phi);
e33=cos(theta).*cos(phi);

a1=e11.*Fxe + e21.*Fye + e31.*Fze;
a2=e12.*Fxe + e22.*Fye + e32.*Fze;
a3=e13.*Fxe + e23.*Fye + e33.*Fze;
a5=Fb*d3.*sin(theta) - Ktheta.*(theta-thetao) - Kthetadot.*thetadot;
a6=-Kpsi.*(psi-psio) - Kpsidot.*psidot;

Thrust=cinv*[a1';a2';a3';a5';a6'];

figure(1)
subplot(321); plot(t,x); xlabel('time (sec)'); ylabel('x (ft)')
subplot(323); plot(t,y); xlabel('time (sec)'); ylabel('y (ft)')
subplot(325); plot(t,z); xlabel('time (sec)'); ylabel('z (ft)')
subplot(322); plot(t,psi); xlabel('time (sec)'); ylabel('psi (rad)')
subplot(324); plot(t,theta); xlabel('time (sec)'); ylabel('theta (rad)')
subplot(326); plot(t,phi); xlabel('time (sec)'); ylabel('phi (rad)')
% orient landscape
print posture.auv3d

figure(2)
subplot(321); plot(t,Thrust(1,:));xlabel('time (sec)');
ylabel('T1 (lbs)');title('Aft Vertical Thruster-T1');
subplot(322); plot(t,Thrust(2,:)); xlabel('time (sec)');
ylabel('T2 (lbs)');title('Fwd Vertical Thruster-T2');
subplot(323); plot(t,Thrust(4,:)); xlabel('time (sec)');
ylabel('T4 (lbs)');title('Aft Lateral Thruster-T4');
subplot(324); plot(t,Thrust(5,:)); xlabel('time (sec)');
ylabel('T5 (lbs)');title('Fwd Lateral Thruster-T5');
subplot(325); plot(t,Thrust(3,:)); xlabel('time (sec)');
ylabel('T3 (lbs)');title('Main Screws Combined-T3');
% orient landscape
print thrusters.auv3d
```

```
% function that to calculate the state variables

function vprime = vd3d(t,v,par)

% Define global variables
global m I g xo yo zo psio thetao phio omega zeta K Kdot mam Ixx Iyy Izz mwam
global Kx Kxdot Ky Kydot Kz Kzdot Kpsi Kpsidot Ktheta Kthetadot Kphi Kphidot

v1dot=[ v(2)];

v2dot=[-Kx/mwam(4)*(v(1)-xo) - Kxdot/mwam(4)*v(2)];

v3dot=[v(4)];

v4dot=[-Kz/mwam(6)*(v(3)-zo) - Kzdot/mwam(6)*v(4)];

v5dot=[v(6)];

v6dot=[-Ktheta/mwam(2)*(v(5)-thetao) - Kthetadot/mwam(2)*v(6)];

v7dot=[v(8)];

v8dot=[-Kpsi/mwam(3)*(v(7)-psio) - Kpsidot/mwam(3)*v(8)];

v9dot=[v(10)];

v10dot=[-Kphi/mwam(1)*(v(9)-phio) - Kphidot/mwam(1)*v(10)];

v11dot=[v(12)];

v12dot=[-Ky/mwam(5)*(v(11)-yo) - Kydot/mwam(5)*v(12)];

vprime=[v1dot;v2dot;v3dot;v4dot;v5dot;v6dot;v7dot;v8dot;v9dot;v10dot;v11dot;v12dot];
```

LIST OF REFERENCES

Cooke, J. M., Zyda, M. J., Pratt, D. R., McGhee, R. B., "NPSNET: Flight Simulation Dynamic Modeling Using Quaternions," *Presence*, Vol.1, No.4, 1992.

DeBitetto, P. A., "Fuzzy Logic for Depth Control of Unmanned Undersea Vehicles," *Proceedings of the Oceanic Engineering Society Symposium on Autonomous Underwater Vehicle Technology*, AUV-94 Cambridge, MA., July 1994.

Dougherty, F., Woolweaver, G., "At Sea Testing of an Unmanned Underwater Vehicle Flight Control System," *Proceedings of the Oceanic Engineering Society Symposium on Autonomous Underwater Vehicle Technology*, AUV-90 Washington, D.C., June 1990.

Frank, A. A., McGhee, R. B., "Some Considerations Relating to the Design of Autopilots for Legged Vehicles," *Journal of Terramechanics*, 1969, Vol.6, No.1, Pergamon Press.

Fujii, T., Ura, T., "Development of Motion Control System for AUV Using Neural Nets," *Proceedings of the Oceanic Engineering Society Symposium on Autonomous Underwater Vehicle Technology*, AUV-90 Washington, D.C., June 1990.

Healey, A. J., Lienard, D., "Multivariable Sliding Mode Control for Autonomous Diving and Steering of Unmanned Underwater Vehicles," *IEEE Journal of Oceanic Engineering,* Vol.18, No.3, July 1993.

Jalving, B., "The NDRE-AUV Flight Control System," *IEEE Journal of Oceanic Engineering*, Vol.19, No.4, October 1994.

Kato, N., "Applications of Fuzzy Algorithm to Guidance and Control of Underwater Vehicles," Workshop on Future Research Directions in Underwater Robotics, 1994, TSI Press, 1995.

McGhee, R. B., Nakano, E., Koyachi, N., Adachi, H., "An Approach to Computer Coordination of Motion for Energy-Efficient Walking Machines," Bulletin of Mechanical Engineering Laboratory, No.43, 1986.

McGhee, R. B., Class Notes on Symbolic Computing, 1994.

Paul, R. P., "Robot Manipulators," MIT Press, Cambridge, MA, 1981.

Thompson, S., "Control Systems: Engineering and Design," John Wiley and Sons, Inc., New York, NY, 1989.

Torsiello, K. A., "Acoustic Positioning of the NPS Autonomous Underwater Vehicle (AUV II) During Hover Conditions," Master's Thesis, Naval Postgraduate School, Monterey, CA, March 1994.

Warner, D. C., "Design, Simulation and Experimental Verification of a Computer Model and Enhanced Position Estimator for the NPS AUV II," Master's Thesis, Naval Postgraduate School, Monterey, CA, December 1991.

Yoerger, D. R., Newman, J. B., Slotine, J. E., "Supervisory Control System for the JASON ROV," *IEEE Journal of Oceanic Engineering*, Vol. OE-11, No.3, July 1986.

Yuh, J., "Underwater Robotic Vehicles: Design and Control," Workshop on Future Research Directions in Underwater Robotics, 1994, TSI Press, 1995.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 52 Naval Postgraduate School Monterey, California 93943-5101	2
3.	Chairman, Code EC Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5121	1
4.	Dr. Robert B. McGhee, Code CS/Mz Department of Computer Science Naval Postgraduate School Monterey, California 93943-5118	2
5.	Dr. Anthony J. Healey, Code ME/Hy Department of Mechanical Engineering Naval Postgraduate School Monterey, California 93943-5108	1
6.	Dr. Harold A. Titus, Code EC/Ts Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5121	1
7.	Dr. Michael B. Matthews 160 Central Avenue Pacific Grove, California 93950	1

8. LT. Juan C. Gonzalez, Jr. 2
1296 Spruance Rd.
Monterey, California 93940

9. Mr. Norman Caplan 1
Environmental and Oceanic Systems Division
National Science Foundation
4201 Wilson Blvd.
Arlington, Virginia 22230